Toll parking library prototype

Java API for a toll parking

Enrico MOLINO

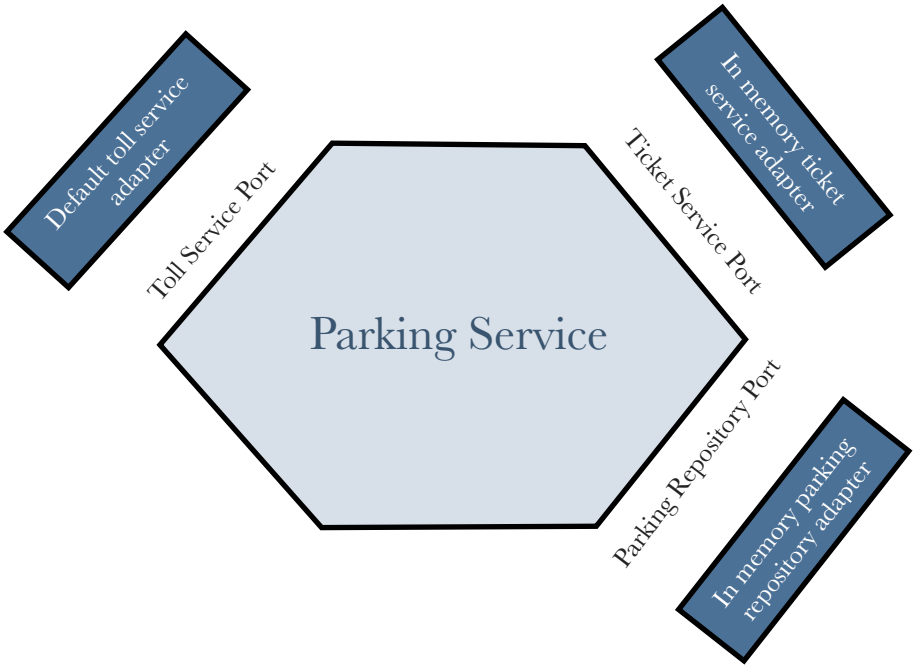
26 May 2019

Introduction

The toll parking library is designed to meet the following specifications:

- Support different parking slot types (standard cars and two different types of electric powered)
- Implement different kinds of pricing policies, that can be added at any time
- Manage the parking slots, guide customers to the right slot, free the slot when the car leaves it, bill the customer

To ensure maximum flexibility, the API design has been inspired by the **Hexagonal Architecture**:



The **application core**, with the business logic, is inside the hexagone and it access to the external world with some **ports**, that are implemented (outside the hexagone) by some **adapters**.

As it is only a prototype, the persistence layer (**Parking Repository**) is implemented in-memory, but it can be easily replaced with a real database

adapter; the same apply for the **Ticket Service Adapter** (it generates a unique parking ticket each time that it is invoked).
The **Toll Service Adapter** works in a different way, as the parking service supports many different toll service adapters in the same time: each parking can have a different toll policy and it can be changed anytime, also at runtime.

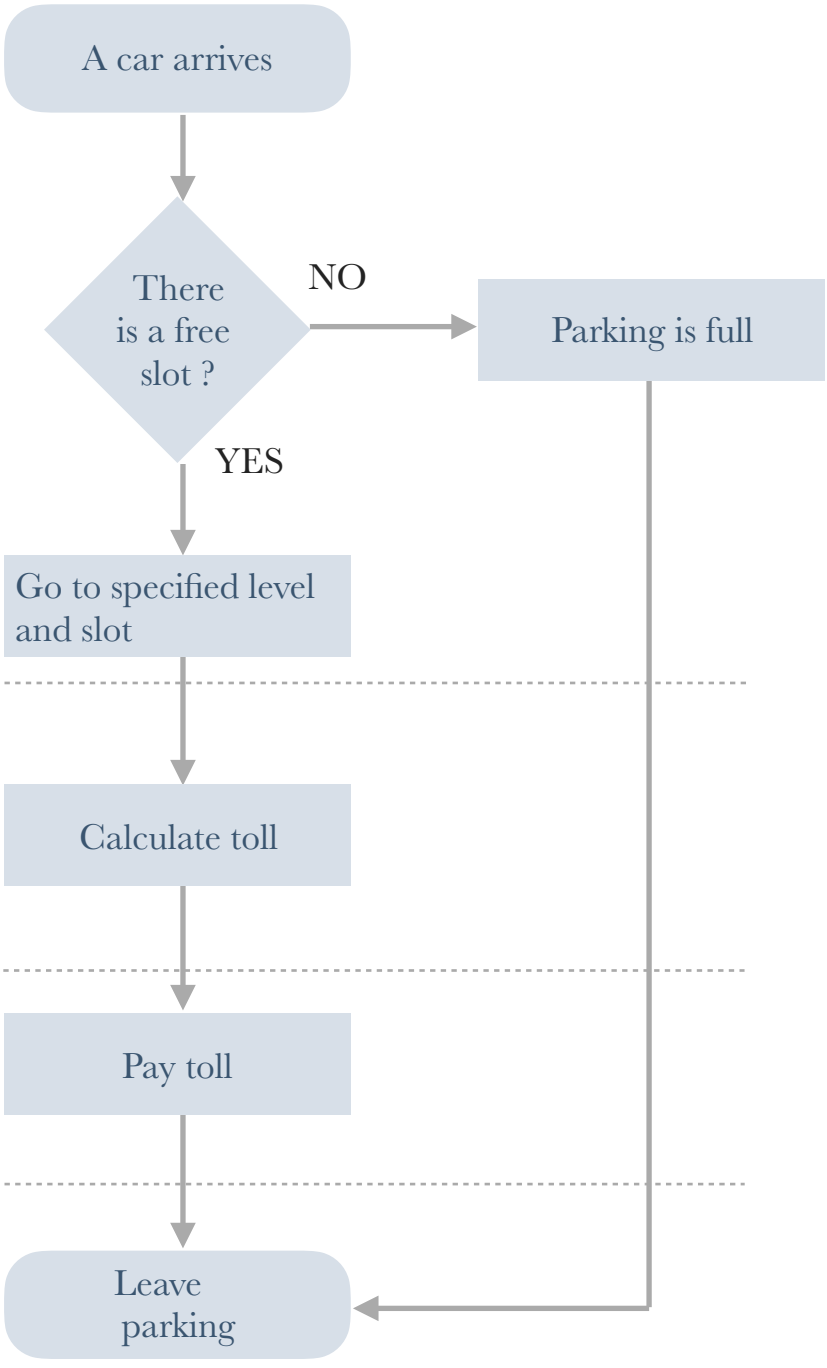
The simplified workflow to use the API is the following:

```
Park arrive(
@NotNull String parkingId,
String licensePlate,
@NotNull ParkingSlotTypeEnum
type,
@NotNull LocalDateTime
arrivalDateTime)
throws ParkingFullException;

The Park object returned
contains the ticket number and
the informations about slot
number and parking level to
park
```

```
BigDecimal calculateToll(
@NotNull String parkingId,
@NotNull Long ticketNumber,
@NotNull LocalDateTime
leaveDateTime)
throws
TicketNotFoundException;

It returns the amount to pay
```

```
void payToll(@NotNull Long
ticketNumber, @NotNull
LocalDateTime
paymentDateTime) throws
TicketNotFoundException;
```

```
void leave(@NotNull Long
ticketNumber, @NotNull
LocalDateTime leaveDateTime)
                    throws
TicketNotFoundException,
TollNotPayedException,
MoreTollToPayException;
```



Pre-requirements and installation

Toll parking library requires:

- **Java 11 or later** (it has been chosen because it is the latest with "Long Term Support" by Oracle) https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html
- **Apache Maven 3.6.1** https://maven.apache.org/download.cgi
- **Git 2.21.0** https://git-scm.com/downloads

The installation of required software it out of the scope of this manual, but you can use install them with default settings.

Installation:

From the terminal, move in the folder where you want install the library and type:

```
git clone https://github.com/enrico77/parking-application.git
```

To compile the Toll Parking Library move in the sub-folder newly created "**parking-application**" (cd parking-application), then type:

```
mvn clean install
```

The application will compile, all JUnit test are ran and also using the plugin Jacoco a report is written in sub-folder **/parking-application/target/site/jacoco/index.html**
The test coverage is >80%.

Use the library

There is a main class, with the only purpose of basic testing. Type under maven target folder (**/parking-application/target**) the following command:

```
java -cp parking-application-0.9.0.jar com.parking.ParkingApp
```

A parking with 115 slots is created, a car try to enter, a ticket is assigned to the car (now the parking has only 114 free slots) then one hour later the car leave the parking, paying 1$:

```
Parking initialized
There are 115 free parking slots
A standard car arrives
The driver has received ticket n.1 and the assigned slot is 2, at
level 1
There are 114 free parking slots
One hour later, the driver use the ticket to calculate the toll
The calculated toll is 1.00$
Pay toll
Exit from parking
There are 115 free parking slots
```

There is a much more detailed JUnit test in: **/parking-application/src/test/java/com/parking/core/ParkingServiceTest.java**
Please take a look at it to have detailed example with all possible API use and all kind of exception you can receive.

To use the library you need instantiate ParkingService:

```
ParkingService parkingService = new ParkingServiceImpl(new
InMemoryParkingRepositoryImpl(), new InMemoryTicketServiceImpl());
```

Please note that the adapters has been injected manually; a final version of this library could use dependency injection, if another framework is used (Spring, Java EE).
The first action is to initialize a parking and related slot, e.g.:

```
Parking parking = new Parking("PK001", "City Center Parking", "unknown
address");
parking.generateMultipleSlots((short) 1, (short) 100, (short) 1,
ParkingSlotTypeEnum.STANDARD);
```

```
parking.generateMultipleSlots((short) 101, (short) 110, (short) 1,
ParkingSlotTypeEnum.ELECTRIC_20KW);
            parking.generateMultipleSlots((short) 111, (short) 115,
(short) 1, ParkingSlotTypeEnum.ELECTRIC_50KW);
```

As you can see, a parking is created with a unique identifier, a name, an address; then 100 parking slot for standard cars, 10 for electric 20KW, 5 for electric 50KW are set.

```
parking.setTollService(new DefaultTollServiceImpl(BigDecimal.valueOf(0d),
BigDecimal.valueOf(1d), BigDecimal.valueOf(1.5d),
BigDecimal.valueOf(2d)));
```

The service used to calculate toll is set to the parking. In this case `DefaultTollServiceImpl` has been used, but you can write your own implementation by implementing the following interface:

```
public interface TollService extends Serializable {

      /**
       *
       * @param arrivalDateTime
       * @param leaveDateTime
       * @param type of parking
       * @return the toll amount
       */
      BigDecimal parkingToll(@NotNull LocalDateTime arrivalDateTime,
@NotNull LocalDateTime leaveDateTime, @NotNull ParkingSlotTypeEnum type);

      /**
       *
       * @return the maximum time that you can stay in the parking without
paying. It is useful to have some time to exit after paying,
        * or also to exit from the parking if you are entered in the
parking by mistake.
       */
      Duration maxDurationAllowedBeforeLeave();
}
```

Finally create the parking with the following command:

```
parkingService.createParking(parking);
```

Now you can use all methods exposed by the interface `ParkingService`, please refers to the examples in JUnit and also the workflow at page 3.

```
public interface ParkingService {

	/**
	 * Get an existing parking by unique id
	 *
	 * @param id
	 * @return
	 */
	Parking getParking(@NotNull String id);

	/**
	 * Create a new parking
	 *
	 * @param parking
	 * @return
	 */
	Parking createParking(@NotNull Parking parking);

	/**
	 * Delete an existing parking
	 *
	 * @param parking
	 */
	void deleteParking(@NotNull Parking parking);

	/**
	 * Count the number of free slots
	 *
	 * @param parkingId
	 * @param type optional, if you don't specify it counts all kind of
slot
	 * @return
	 */
	Long countFreeSlots(@NotNull String parkingId, ParkingSlotTypeEnum
type);

	/**
	 * Call this method when entering in the parking
	 *
	 *
	 * @param parkingId
	 * @param licensePlate (optional, it can be null)
	 * @param type
	 *            (standard/electric 20KW/electric 50KW)
	 * @param arrivalDateTime
	 *            (provide the current local date time)
	 * @return
	 * @throws ParkingFullException
	 *             (thrown if the parking is full)
	 */
	Park arrive(@NotNull String parkingId, String licensePlate, @NotNull
ParkingSlotTypeEnum type, @NotNull LocalDateTime arrivalDateTime)
			throws ParkingFullException;
```

```
	/**
	 * Calculate the parking toll
	 *
	 * @param parkingId
	 * @param ticketNumber
	 * @param leaveDateTime
	 * @return
	 * @throws TicketNotFoundException
	 */
	BigDecimal calculateToll(@NotNull String parkingId, @NotNull Long
ticketNumber, @NotNull LocalDateTime leaveDateTime)
			throws TicketNotFoundException;

	/**
	 * Inform the system that the toll has been payed
	 *
	 * @throws TicketNotFoundException
	 */
	void payToll(@NotNull Long ticketNumber, @NotNull LocalDateTime
paymentDateTime) throws TicketNotFoundException;

	/**
	 * Call this method before authorize the driver to quit the parking
	 *
	 * @param ticketNumber
	 * @param leaveDateTime
	 * @throws TicketNotFoundException
	 * @throws TollNotPayedException
	 * @throws MoreTollToPayException
	 */
	void leave(@NotNull Long ticketNumber, @NotNull LocalDateTime
leaveDateTime)
			throws TicketNotFoundException, TollNotPayedException,
MoreTollToPayException;

	/**
	 * If the driver has lost the ticket, the ticket number can be
retrieved using
	 * the license plate. However it is not always possible, and a
camera with OCR
	 * is needed in the parking, that is optional
	 *
	 * @param parkingId
	 * @param licensePlate
	 * @return
	 * @throws DuplicatePlateException
	 * @throws PlateNotFoundException
	 */
	Long retrieveTicketNumber(@NotNull String parkingId, @NotNull String
licensePlate)
			throws DuplicatePlateException, PlateNotFoundException;
```

```
      /**
       * Update toll service for a specific parking
       *
       * @param parkingId
       * @param tollService
       */
      void udateTollService(@NotNull String parkingId, @NotNull
TollService tollService);
}
```