

Type de chambre d'hôtel : énumération RoomType

Tâche 1 : Compléter l'énumération RoomType dans le fichier `src/main/java/RoomType.java` qui respecte le diagramme ci-dessous.

```
/**
 * An enum to model a type of room in a hotel.
 *
 * A type of room has a maximum occupancy (the largest number of people who can legally
 * sleep in a room of this type) and a price per night.
 */
public enum RoomType {
    /**
     * A type of room corresponding to a single room with a price of 60 and
     * a maximum occupancy of 1.
     */
    SINGLE(60,1),
    /**
     * A type of room corresponding to a double room with a price of 90 and
     * a maximum occupancy of 2.
     */
    DOUBLE(90,2),
    /**
     * A type of room corresponding to a double-double room with a price of 150
     * and a maximum occupancy of 4.
     */
    DOUBLE_DOUBLE(150, 4),
    /**
     * A type of room corresponding to a suite room with a price of 300 and
     * a maximum occupancy of 4.
     */
    SUIT(300, 4);

    private final int price;
    private final int maximumOccupancy;

    /**
     * create a type of room with the specified price and maximum occupancy
     * @param price the price of the type of room
     * @param maximumOccupancy the maximum occupancy of this type of room
     */
    RoomType(int price, int maximumOccupancy) {
        this.price = price;
    }
}
```

```

        this.maximumOccupancy = maximumOccupancy;
    }
    /**
     * return the price of this type of room
     * @return the price of this type of room
     */
    public int getPrice() {
        return price;
    }
    /**
     * return the maximum occupancy of this type of room
     * @return the maximum occupancy of this type of room
     */
    public int getMaximumOccupancy() {
        return maximumOccupancy;
    }
}

```

Classes de test RoomType

Tâche 2 : Compléter la classe de test `RoomTypeTest` dans le fichier `src/test/java/RoomTypeTest.java` pour qu'elle contienne des tests pour vérifier le bon fonctionnement de la valeur `DOUBLE_DOUBLE` de `RoomType`.

```

import org.junit.jupiter.api.Test;

import static org.assertj.core.api.Assertions.assertThat;

public class RoomTypeTest {
    @Test
    void testSinglePrice(){
        assertThat(RoomType.SINGLE.getPrice()).isEqualTo(60);
    }
    @Test
    void testSingleMaximumOccupancy(){
        assertThat(RoomType.SINGLE.getMaximumOccupancy()).isEqualTo(1);
    }
    @Test
    void testSingleToString(){
        assertThat(RoomType.SINGLE.toString()).isEqualTo("SINGLE");
    }
    @Test
    void testDoubleDoublePrice(){
        assertThat(RoomType.DOUBLE_DOUBLE.getPrice()).isEqualTo(150);
    }
    @Test

```

```

void testDoubleDoubleMaximumOccupancy(){
    assertThat(RoomType.DOUBLE_DOUBLE.getMaximumOccupancy()).isEqualTo(4);
}
@Test
void testDoubleDoubleToString(){
    assertThat(RoomType.DOUBLE_DOUBLE.toString()).isEqualTo("DOUBLE_DOUBLE");
}
}

```

Chambre d'hôtel : classe Room

Tâche 3 : Complétez la classe Room dans le fichier `src/main/java/Room.java` qui respecte le diagramme ci-dessous.

```

/**
 * A class to model Room in a hotel.
 *
 * A room has a unique number and can be rented or not.
 */
public class Room {
    private final int number;
    private final RoomType roomType;
    private boolean isRented = false;

    /**
     * create a room with given number and room type, initially a room is not rented
     * @param number number of this room
     */
    public Room(RoomType roomType, int number) {
        this.roomType = roomType;
        this.number = number;
    }

    /**
     * return this room number
     * @return this room number
     */
    public int getNumber() {
        return number;
    }

    /**
     * return the type of this room
     * @return the type of this room
     */
    public RoomType getRoomType() {
        return roomType;
    }
}

```

```

    * return the price of this room
    * @return the price of this room
    */
public int getPrice() {
    return roomType.getPrice();
}

/**
    * return the maximum occupancy of this room
    * @return the maximum occupancy of this room
    */
int getMaximumOccupancy(){
    return roomType.getMaximumOccupancy();
}

/**
    * return <code>true</code> if and only if this room is rented
    * @return <code>true</code> if and only if this room is rented
    */
public boolean isRented() {
    return isRented;
}

/**
    * rent this room if it was not rented, returning {@code true} if the room was not rented
    *
    * @return {@code true} if the room was not rented and {@code false} otherwise
    */
public boolean rent() {
    if(isRented) return false;
    isRented = true;
    return true;
}

/**
    * free this room if it was rented, returning {@code true} if the room was rented
    *
    * @return {@code true} if the room was rented and {@code false} otherwise
    */
public boolean leave() {
    if(!isRented) return false;
    isRented = false;
    return true;
}

/**
    * return a string representation of the room containing the room in the following format :
    * ZZ room, number XX, price YY$ where ZZ is the type of this room, XX is the number of this
    * room and YY is the price of this room.
    *
    * @return a string representation of this room

```

```

    */
    public String toString() {
        return roomType + " room, number " + number + ", price " + getPrice() + "$";
    }

    /**
     * Determines whether two rooms are equal. Two instances of Room are equal if their number
     * are the same.
     *
     * @return {@code true} if the object to be compared is an instance of Room and has the
     * same number; false otherwise.
     * @see java.lang.Object#equals(java.lang.Object)
     */
    public boolean equals(Object o) {
        if(! (o instanceof Room))
            return false;
        Room room = (Room) o;
        return room.number == number;
    }
}

```

Classes de test RoomTest

Tâche 4 : Compléter la classe de test RoomTest dans le fichier `src/test/java/RoomTest.java` pour qu'elle contienne des tests pour vérifier le bon fonctionnement des méthodes `equals`, `free` et `rent` de la classe Room.

```

import org.junit.jupiter.api.Test;
import static org.assertj.core.api.Assertions.*;

public class RoomTest {
    @Test
    void testToStringContains(){
        Room room = new Room(RoomType.SINGLE, 666);
        assertThat(room.toString()).contains("666")
            .contains(RoomType.SINGLE.toString())
            .contains(String.valueOf(RoomType.SINGLE.getPrice()))
            .contains(String.valueOf(room.getPrice()));
    }
    @Test
    void testToStringStrictFormat(){
        Room room = new Room(RoomType.SINGLE, 12);
        assertThat(room.toString()).isEqualTo("%s room, number %d, price %d$",
            RoomType.SINGLE, room.getNumber(), RoomType.SINGLE.getPrice());
    }
    @Test

```

```

void testEquals(){
    Room room1 = new Room(RoomType.SINGLE, 1);
    Room room1Bis = new Room(RoomType.SINGLE, 1);
    Room room1Ter = new Room(RoomType.DOUBLE, 1);
    Room room11 = new Room(RoomType.SINGLE, 11);
    assertThat(room1).isEqualTo(room1)
        .isEqualTo(room1Bis)
        .isEqualTo(room1Ter)
        .isNotEqualTo(room11);
}

@Test
void testRentFreeRoom(){
    Room room = new Room(RoomType.SINGLE, 666);
    assertThat(room.isRented()).isFalse();
    assertThat(room.rent()).isTrue();
    assertThat(room.isRented()).isTrue();
}

@Test
void testRentAlreadyRentedRoom(){
    Room room = new Room(RoomType.SINGLE, 666);
    room.rent();
    assertThat(room.isRented()).isTrue();
    assertThat(room.rent()).isFalse();
    assertThat(room.isRented()).isTrue();
}

@Test
void testFreeRentedRoom(){
    Room room = new Room(RoomType.SINGLE, 666);
    room.rent();
    assertThat(room.isRented()).isTrue();
    assertThat(room.leave()).isTrue();
    assertThat(room.isRented()).isFalse();
}

@Test
void testFreeAlreadyFreeRoom(){
    Room room = new Room(RoomType.SINGLE, 666);
    assertThat(room.isRented()).isFalse();
    assertThat(room.leave()).isFalse();
    assertThat(room.isRented()).isFalse();
}
}

```

Hôtel : classe Hotel

Une chambre d'hôtel sera représentée par une instance de la classe Room.

Tâche 5 : Complétez la classe `Hotel` dans le fichier `src/main/java/Hotel.java` qui respecte le diagramme ci-dessous.

```
import java.util.List;
import java.util.ArrayList;

/**
 * A class to model a hotel.
 *
 * A Hotel has a name and a list of rooms.
 */
public class Hotel {
    private final String name;
    private final List<Room> rooms = new ArrayList<>();

    /**
     * create a Hotel with the given name and no room
     * @param name this hotel name
     */
    public Hotel(String name) {
        this.name = name;
    }

    /**
     * return this hotel name
     * @return this hotel name
     */
    public String getName() {
        return name;
    }

    /**
     * return the number of rooms of this hotel
     * @return the number of rooms of this hotel
     */
    public int getNumberOfRooms() {
        return rooms.size();
    }

    /**
     * provide the room corresponding to the given number, return {@code null}
     * if there is no room in the hotel with that number.
     *
     * @param number number of the room
     * @return the room with given number if it exists and {@code null} otherwise
     */
    public Room getRoom(int number) {
        for(Room room : rooms){
```

```

        if(room.getNumber() == number)
            return room;
    }
    return null;
}

/**
 * return the revenue of the hotel, i.e., the sum of the prices of the rented rooms
 *
 * @return the sum of the prices of the rented rooms
 */
public int getRevenue() {
    int sum = 0;
    for(Room room : rooms){
        if(room.isRented())
            sum += room.getPrice();
    }
    return sum;
}

/**
 * return the total occupancy of the hotel, i.e., the sum of the maximum occupancies
 * of its rooms
 *
 * @return the sum of the maximum occupancies of the rooms
 */
public int getTotalOccupancy() {
    int sum = 0;
    for(Room room : rooms){
        sum += room.getMaximumOccupancy();
    }
    return sum;
}

/**
 * add the specified room if there is not already a room in this hotel with the same
 * room number
 *
 * @param room the room to be added
 * @return {@code true} if the room was added as a result of the call (there was no
 * room with the same number)
 */
public boolean addRoom(Room room) {
    if (containsRoomWithSameNumberAs(room))
        return false;
    rooms.add(room);
    return true;
}

private boolean containsRoomWithSameNumberAs(Room room) {

```

```

    return getRoom(room.getNumber()) != null;
}

/**
 * add each room in the specified list of rooms if there is not already a room in
 * this hotel with the same
 * room number
 *
 * @param rooms the rooms to be added
 */
public void addRooms(List<Room> rooms) {
    for(Room room : rooms)
        addRoom(room);
}

/**
 * rent the room of the hotel corresponding to the specified room number if it was
 * not rented, return {@code true} if such a room exists and was not rented
 *
 * @param number the number of the room to be rented
 * @return {@code true} if the room corresponding to the room number exists and
 * was not rented and {@code false} otherwise
 */
public boolean rentRoom(int number) {
    Room room = getRoom(number);
    if(room == null)
        return false;
    return room.rent();
}

/**
 * leave the room of the hotel corresponding to the specified room number if it
 * was rented, return {@code true} if such a room exists and was rented
 *
 * @param number the number of the room to be left
 * @return {@code true} if the room corresponding to the room number exists and
 * was rented and {@code false} otherwise
 */
public boolean leaveRoom(int number) {
    Room room = getRoom(number);
    if(room == null)
        return false;
    return room.leave();
}

/**
 * return a non rented room corresponding to the specified type of room if such a

```

```

    * room exists and {@code null} otherwise
    *
    * @param roomType the desired type of room
    * @return a free room of the desired type or {@code null} if there is no such room
    */
    public Room getFreeRoom(RoomType roomType) {
        for(Room room : rooms)
            if(room.getRoomType() == roomType && !room.isRented())
                return room;
        return null;
    }
    /**
    * print the hotel name followed by the string representations of the rooms
    */
    public void print(){
        System.out.println(getName());
        for (Room room : rooms) {
            System.out.println(room);
        }
    }
}

```

Hôtel : classe HotelTest (bonus)

Tâche 6 : Compléter la classe de test `HotelTest` dans le fichier `src/test/java/HotelTest.java` pour qu'elle contienne des tests pour vérifier le bon fonctionnement des méthodes `getTotalOccupancy`, `getNumberOfRooms`, `getRevenue`, `addRooms` et `getFreeRoom` de la classe `Hotel`.

```

import org.junit.jupiter.api.Test;

import java.util.ArrayList;
import java.util.List;

import static org.assertj.core.api.Assertions.*;

public class HotelTest {
    @Test
    void testPrint_whenHotelHasNoRoom() {
        Hotel hotel = new Hotel("Infogrames hotel");
        StandardOutputSandbox standardOutputSandbox = new StandardOutputSandbox(hotel::print);
        String expectedOutput = hotel.getName() + StandardOutputSandbox.NEW_LINE;
        standardOutputSandbox.run();
        assertThat(standardOutputSandbox.getProducedOutput()).isEqualTo(expectedOutput);
    }
    @Test

```

```

void testPrint_whenHotelContainsRooms() {
    Hotel hotel = new Hotel("Davilex hotel");
    Room room1 = new Room(RoomType.SINGLE, 1);
    Room room4 = new Room(RoomType.SINGLE, 4);
    hotel.addRooms(List.of(room1, room4));
    assertThat(hotel.getNumberOfRooms()).isEqualTo(2);
    assertThat(hotel.getRoom(1)).isEqualTo(room1);
    assertThat(hotel.getRoom(4)).isEqualTo(room4);
    StandardOutputSandbox standardOutputSandbox = new StandardOutputSandbox(hotel::print);
    String expectedOutput = hotel.getName() + StandardOutputSandbox.NEW_LINE
        + room1 + StandardOutputSandbox.NEW_LINE
        + room4 + StandardOutputSandbox.NEW_LINE;
    standardOutputSandbox.run();
    assertThat(standardOutputSandbox.getProducedOutput()).isEqualTo(expectedOutput);
}

@Test
void testAddRoom(){
    Hotel hotel = new Hotel("Davilex hotel");
    Room room1 = new Room(RoomType.SINGLE, 1);
    Room room1Bis = new Room(RoomType.SINGLE, 1);
    assertThat(hotel.addRoom(room1)).isTrue();
    assertThat(hotel.getRoom(1)).isEqualTo(room1);
    assertThat(hotel.addRoom(room1Bis)).isFalse();
    Room room4 = new Room(RoomType.SINGLE, 4);
    assertThat(hotel.addRoom(room4)).isTrue();
    assertThat(hotel.getRoom(4)).isEqualTo(room4);
}

@Test
void testGetRoom(){
    Hotel hotel = new Hotel("Davilex hotel");
    int[] roomNumbers = {1, 2, 4};
    Room[] rooms = new Room[roomNumbers.length];
    for(int index = 0; index< rooms.length; index++){
        rooms[index] = new Room(RoomType.SINGLE, roomNumbers[index]);
        hotel.addRoom(rooms[index]);
    }
    assertThat(hotel.getNumberOfRooms()).isEqualTo(rooms.length);
    for(int index = 0; index< rooms.length; index++){
        assertThat(hotel.getRoom(roomNumbers[index])).isEqualTo(rooms[index]);
    }
}

@Test
void testLeaveFreeRoom(){
    Hotel hotel = new Hotel("Davilex hotel");
    hotel.addRoom(new Room(RoomType.SINGLE, 1));
}

```

```

    assertThat(hotel.getRoom(1).isRented()).isFalse();
    assertThat(hotel.leaveRoom(1)).isFalse();
    assertThat(hotel.getRoom(1).isRented()).isFalse();
}

@Test
void testLeaveRentedRoom(){
    Hotel hotel = new Hotel("Davilex hotel");
    Room room4 = new Room(RoomType.SINGLE, 4);
    room4.rent();
    hotel.addRoom(room4);
    assertThat(hotel.getRoom(4).isRented()).isTrue();
    assertThat(hotel.leaveRoom(4)).isTrue();
    assertThat(hotel.getRoom(4).isRented()).isFalse();
}

@Test
void testRentFreeRoom(){
    Hotel hotel = new Hotel("Davilex hotel");
    hotel.addRoom(new Room(RoomType.SINGLE, 1));
    assertThat(hotel.getRoom(1).isRented()).isFalse();
    assertThat(hotel.rentRoom(1)).isTrue();
    assertThat(hotel.getRoom(1).isRented()).isTrue();
}

@Test
void testRentRentedRoom(){
    Hotel hotel = new Hotel("Davilex hotel");
    Room room4 = new Room(RoomType.SINGLE, 4);
    room4.rent();
    hotel.addRoom(room4);
    assertThat(hotel.getRoom(4).isRented()).isTrue();
    assertThat(hotel.rentRoom(4)).isFalse();
    assertThat(hotel.getRoom(4).isRented()).isTrue();
}

@Test
void testGetTotalOccupancy(){
    Hotel hotel = new Hotel("Davilex hotel");
    assertThat(hotel.getTotalOccupancy()).isEqualTo(0);
    hotel.addRoom(new Room(RoomType.SINGLE, 1));
    assertThat(hotel.getTotalOccupancy()).isEqualTo(1);
    hotel.addRoom(new Room(RoomType.DOUBLE, 4));
    assertThat(hotel.getTotalOccupancy()).isEqualTo(3);
}

@Test
void testGetNumberOfRooms(){
    Hotel hotel = new Hotel("Davilex hotel");
    assertThat(hotel.getNumberOfRooms()).isEqualTo(0);
    hotel.addRoom(new Room(RoomType.SINGLE, 1));
    assertThat(hotel.getNumberOfRooms()).isEqualTo(1);
}

```

```

    hotel.addRoom(new Room(RoomType.DOUBLE, 4));
    assertThat(hotel.getNumberOfRooms()).isEqualTo(2);
}

@Test
void testGetRevenue(){
    Hotel hotel = new Hotel("Davilex hotel");
    assertThat(hotel.getNumberOfRooms()).isEqualTo(0);
    assertThat(hotel.getRevenue()).isEqualTo(0);
    Room room1 = new Room(RoomType.SINGLE, 1);
    hotel.addRoom(room1);
    assertThat(hotel.getNumberOfRooms()).isEqualTo(1);
    assertThat(hotel.getRevenue()).isEqualTo(0);
    room1.rent();
    assertThat(hotel.getRevenue()).isEqualTo(room1.getPrice());
    Room room4 = new Room(RoomType.DOUBLE, 4);
    hotel.addRoom(room4);
    assertThat(hotel.getNumberOfRooms()).isEqualTo(2);
    assertThat(hotel.getRevenue()).isEqualTo(room1.getPrice());
    room4.rent();
    assertThat(hotel.getRevenue()).isEqualTo(room1.getPrice() + room4.getPrice());
}

@Test
void testAddRooms_onEmptyHotel(){
    Hotel hotel = new Hotel("Davilex hotel");
    assertThat(hotel.getNumberOfRooms()).isEqualTo(0);
    Room room1 = new Room(RoomType.SINGLE, 1);
    Room room2 = new Room(RoomType.SUITE, 2);
    Room room4 = new Room(RoomType.DOUBLE, 4);
    List<Room> rooms = new ArrayList<>();
    rooms.add(room1);
    rooms.add(room2);
    rooms.add(room4);
    hotel.addRooms(rooms);
    assertThat(hotel.getNumberOfRooms()).isEqualTo(3);
}

@Test
void testAddRooms_onHotelWithRooms(){
    Hotel hotel = new Hotel("Davilex hotel");
    assertThat(hotel.getNumberOfRooms()).isEqualTo(0);
    Room room1 = new Room(RoomType.SINGLE, 1);
    Room room1Bis = new Room(RoomType.DOUBLE, 1);
    Room room2 = new Room(RoomType.SUITE, 2);
    Room room4 = new Room(RoomType.DOUBLE, 4);
    Room room4Bis = new Room(RoomType.DOUBLE_DOUBLE, 4);
    hotel.addRoom(room1);
    hotel.addRoom(room4);
    assertThat(hotel.getNumberOfRooms()).isEqualTo(2);
}

```

```

List<Room> rooms = new ArrayList<>();
rooms.add(room1Bis);
rooms.add(room2);
rooms.add(room4Bis);
hotel.addRooms(rooms);
assertThat(hotel.getNumberOfRooms()).isEqualTo(3);
assertThat(hotel.getRoom(1).getRoomType()).isEqualTo(RoomType.SINGLE);
assertThat(hotel.getRoom(4).getRoomType()).isEqualTo(RoomType.DOUBLE);
}

@Test
void testGetFreeRoom_onEmptyHotel(){
    Hotel hotel = new Hotel("Davilex hotel");
    assertThat(hotel.getFreeRoom(RoomType.SINGLE)).isNull();
    assertThat(hotel.getFreeRoom(RoomType.DOUBLE)).isNull();
    assertThat(hotel.getFreeRoom(RoomType.SUITE)).isNull();
    assertThat(hotel.getFreeRoom(RoomType.DOUBLE_DOUBLE)).isNull();
}

@Test
void testGetFreeRoom_onHotelWithOneFreeRoom(){
    Hotel hotel = new Hotel("Davilex hotel");
    Room room = new Room(RoomType.SINGLE, 1);
    hotel.addRoom(room);
    assertThat(hotel.getNumberOfRooms()).isEqualTo(1);
    assertThat(hotel.getFreeRoom(RoomType.SINGLE)).isEqualTo(room);
    assertThat(hotel.getFreeRoom(RoomType.DOUBLE)).isNull();
    assertThat(hotel.getFreeRoom(RoomType.SUITE)).isNull();
    assertThat(hotel.getFreeRoom(RoomType.DOUBLE_DOUBLE)).isNull();
}

@Test
void testGetFreeRoom_onHotelWithTwoFreeRooms(){
    Hotel hotel = new Hotel("Davilex hotel");
    Room room1 = new Room(RoomType.SUITE, 1);
    hotel.addRoom(room1);
    Room room4 = new Room(RoomType.DOUBLE, 4);
    hotel.addRoom(room4);
    assertThat(hotel.getNumberOfRooms()).isEqualTo(2);
    assertThat(hotel.getFreeRoom(RoomType.SUITE)).isEqualTo(room1);
    assertThat(hotel.getFreeRoom(RoomType.DOUBLE)).isEqualTo(room4);
    assertThat(hotel.getFreeRoom(RoomType.SINGLE)).isNull();
    assertThat(hotel.getFreeRoom(RoomType.DOUBLE_DOUBLE)).isNull();
}

@Test
void testGetFreeRoom_onHotelWithTwoRentedRoomsAndOneFreeRoom(){
    Hotel hotel = new Hotel("Davilex hotel");
    Room room1 = new Room(RoomType.SUITE, 1);
    room1.rent();
    assertThat(room1.isRented()).isTrue();
}

```

```
hotel.addRoom(room1);
Room room2 = new Room(RoomType.DOUBLE_DOUBLE, 2);
hotel.addRoom(room2);
Room room4 = new Room(RoomType.DOUBLE, 4);
hotel.addRoom(room4);
room4.rent();
assertThat(room4.isRented()).isTrue();
assertThat(hotel.getNumberOfRooms()).isEqualTo(3);
assertThat(hotel.getFreeRoom(RoomType.SUITE)).isNull();
assertThat(hotel.getFreeRoom(RoomType.DOUBLE)).isNull();
assertThat(hotel.getFreeRoom(RoomType.SUITE)).isNull();
assertThat(hotel.getFreeRoom(RoomType.DOUBLE_DOUBLE)).isEqualTo(room2);
}
}
```