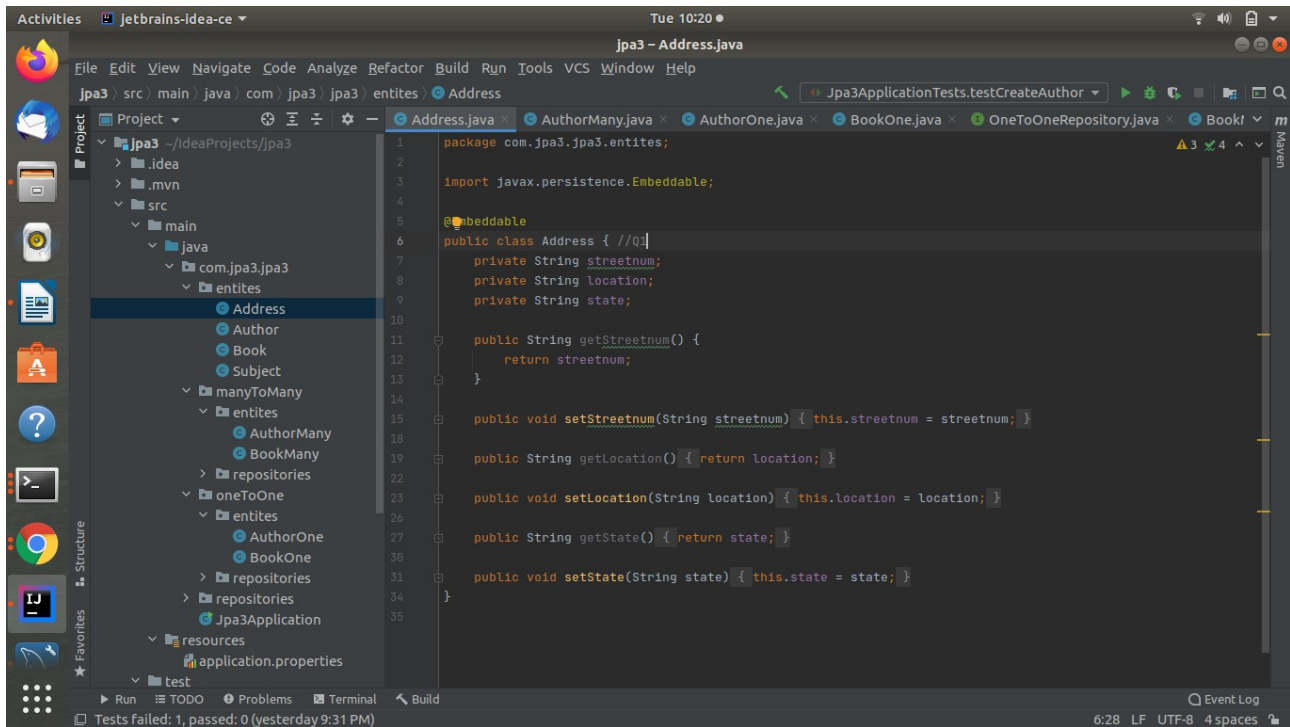
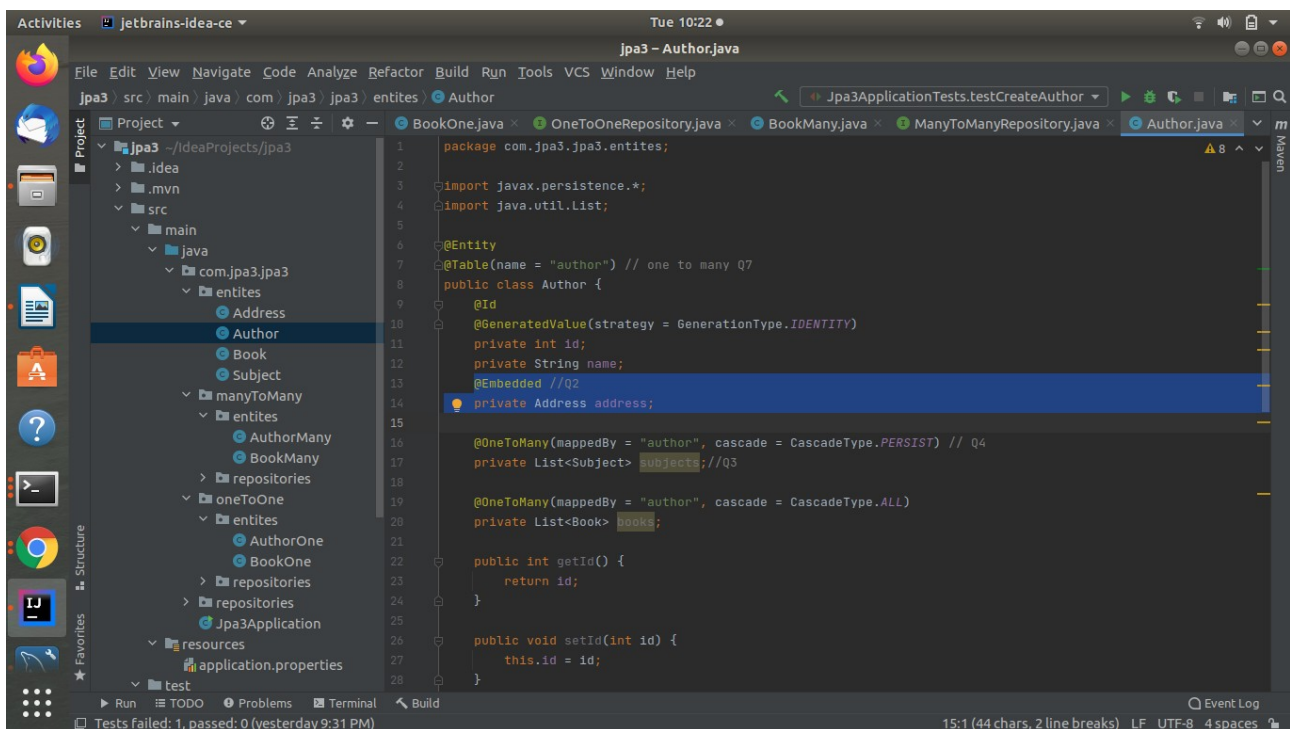


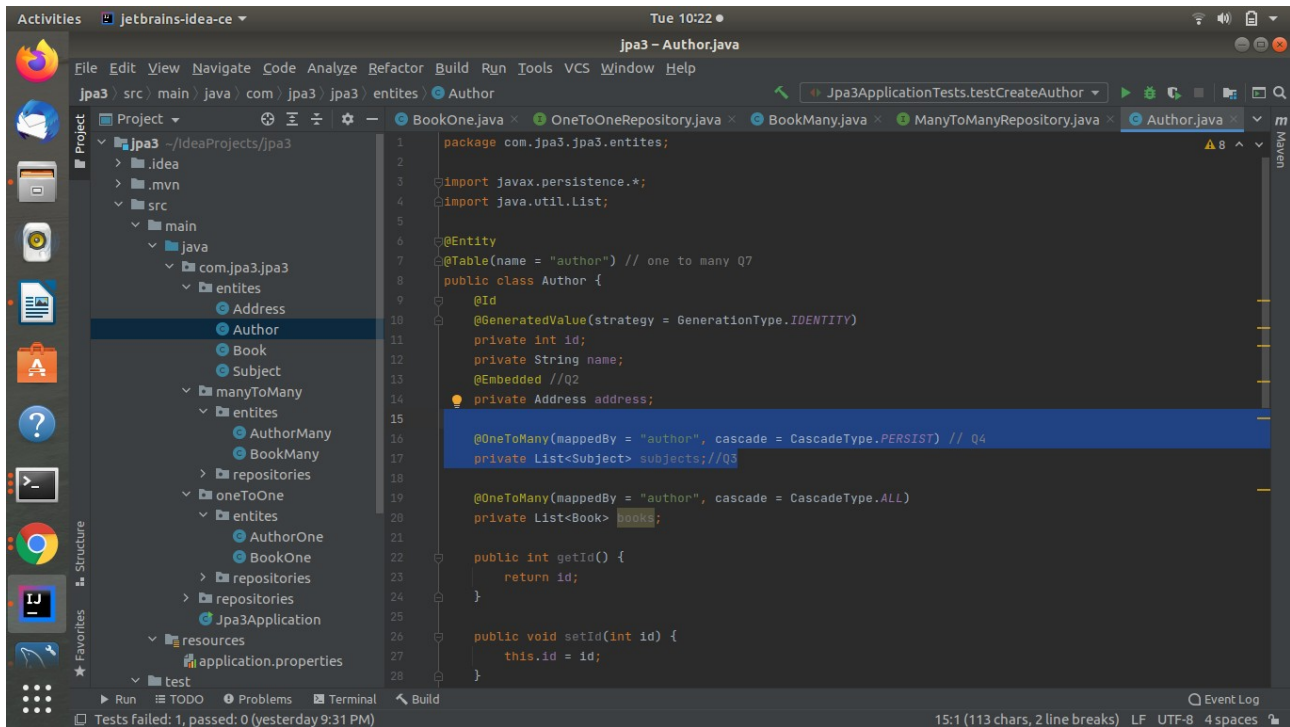
1. Create a class Address for Author with instance variables streetNumber, location, State.



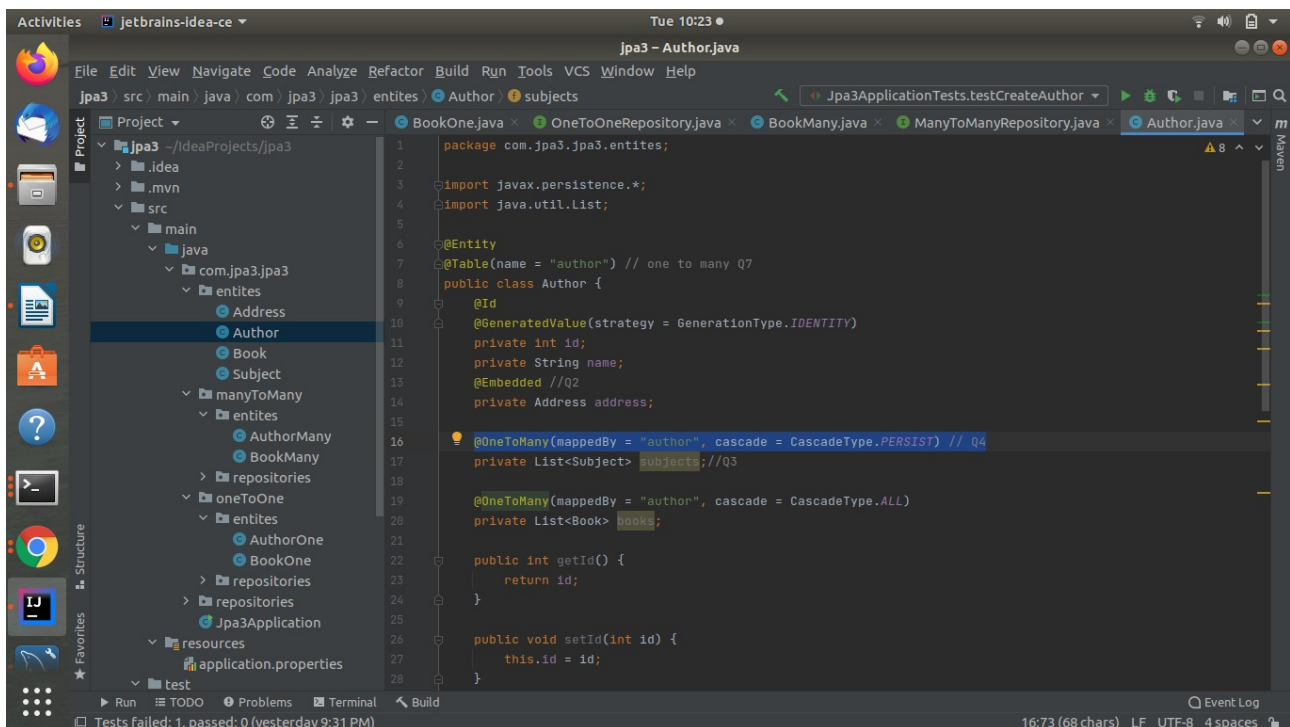
2. Create instance variable of Address class inside Author class and save it as embedded object.



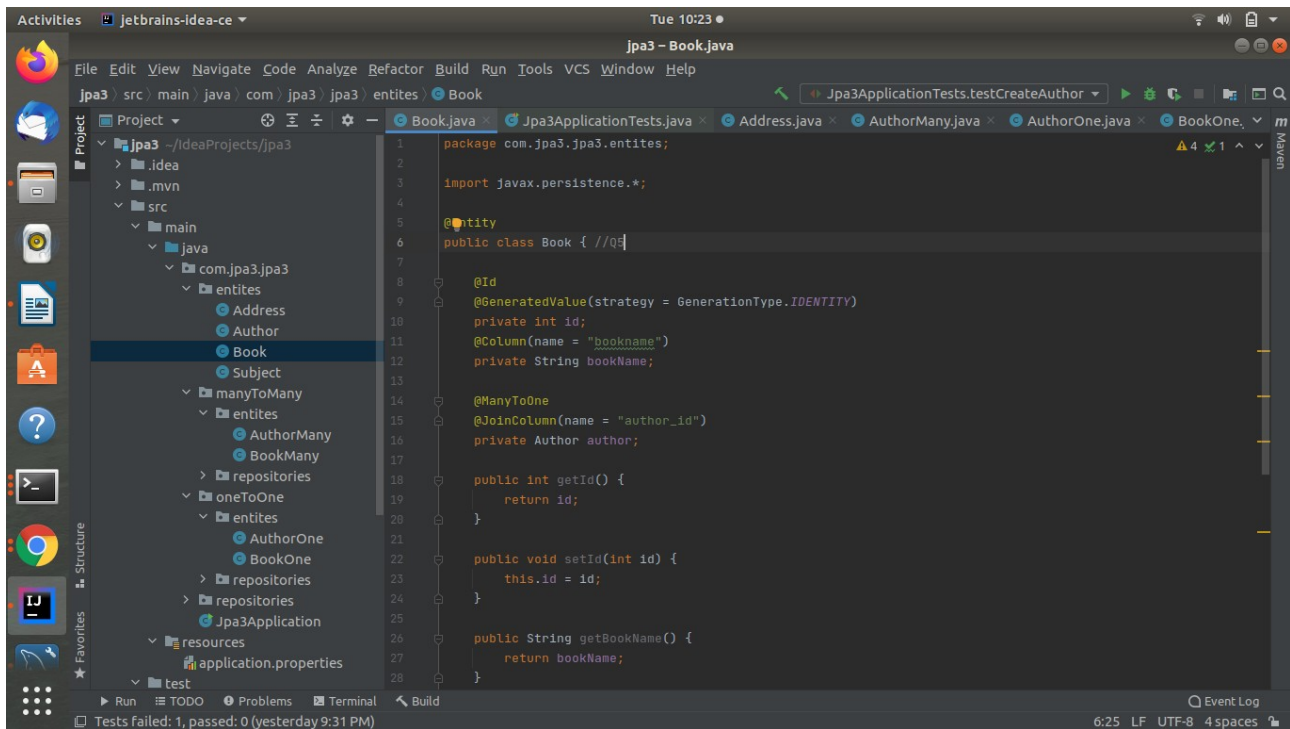
3. Introduce a List of subjects for author.



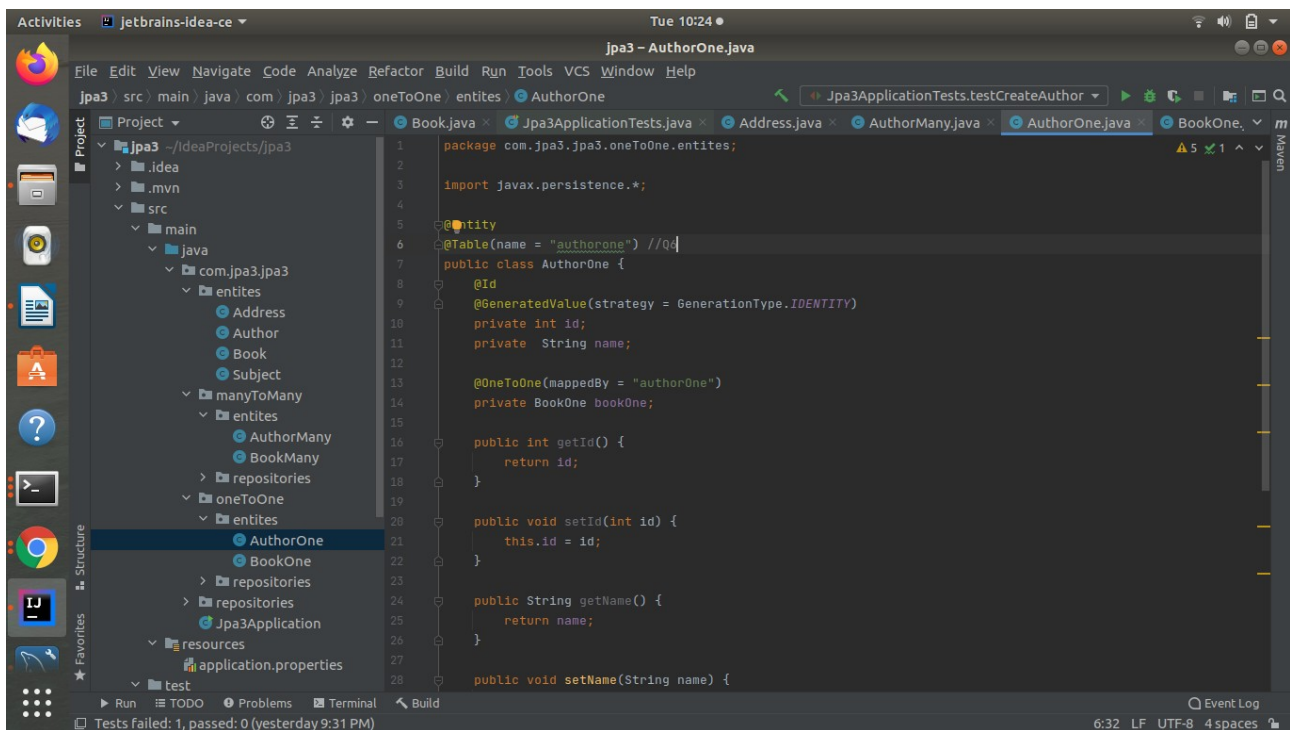
4. Persist 3 subjects for each author.

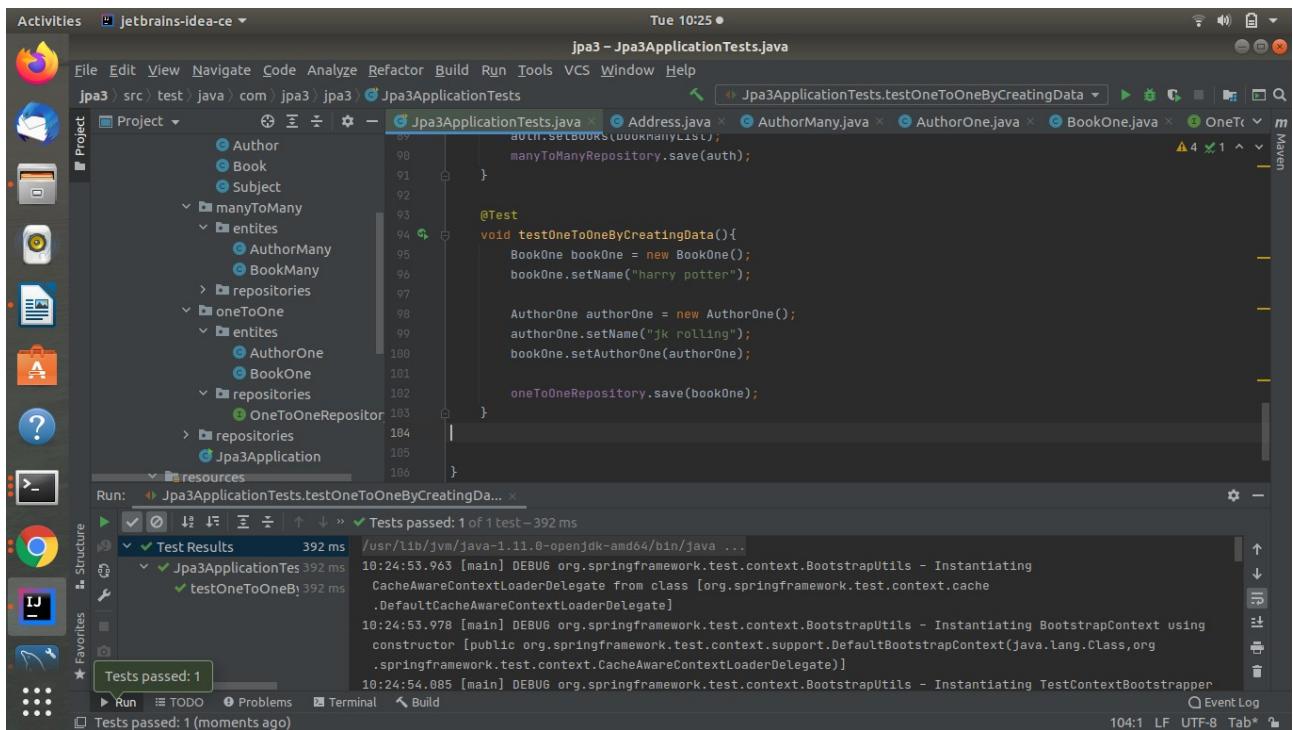
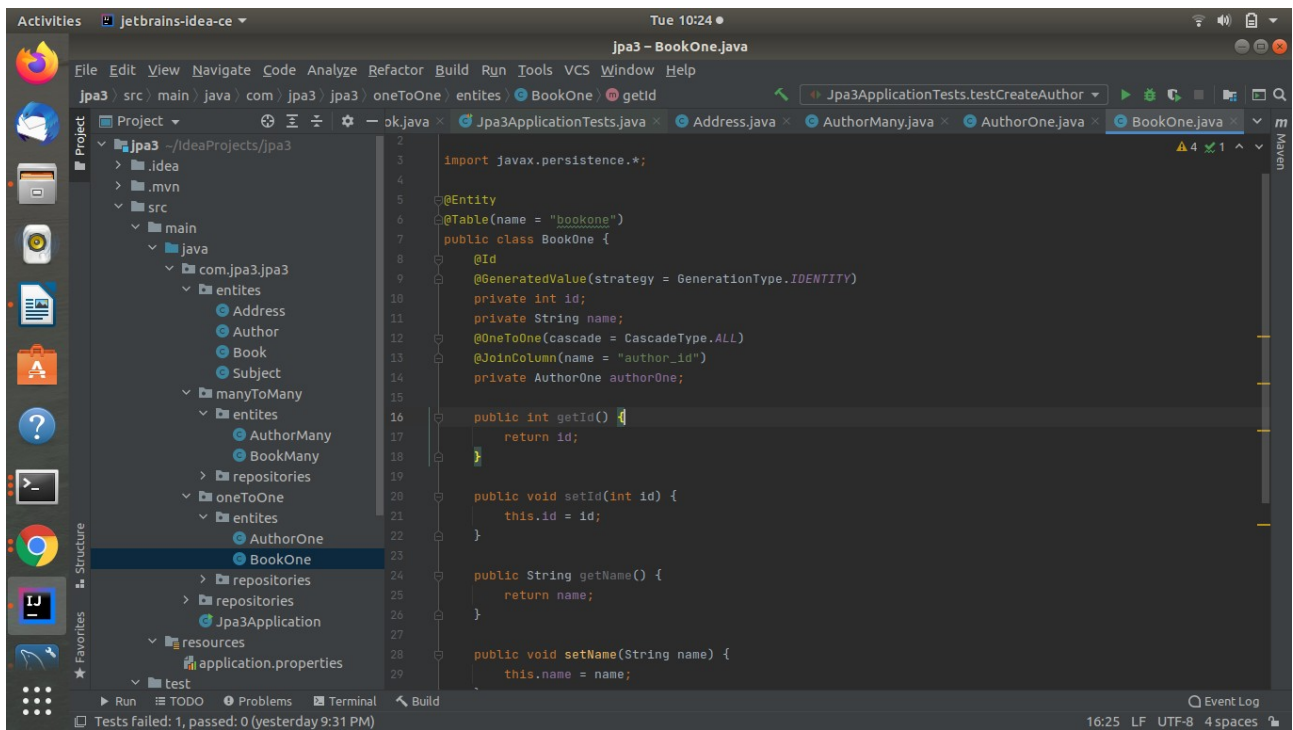


5. Create an Entity book with an instance variable bookName.

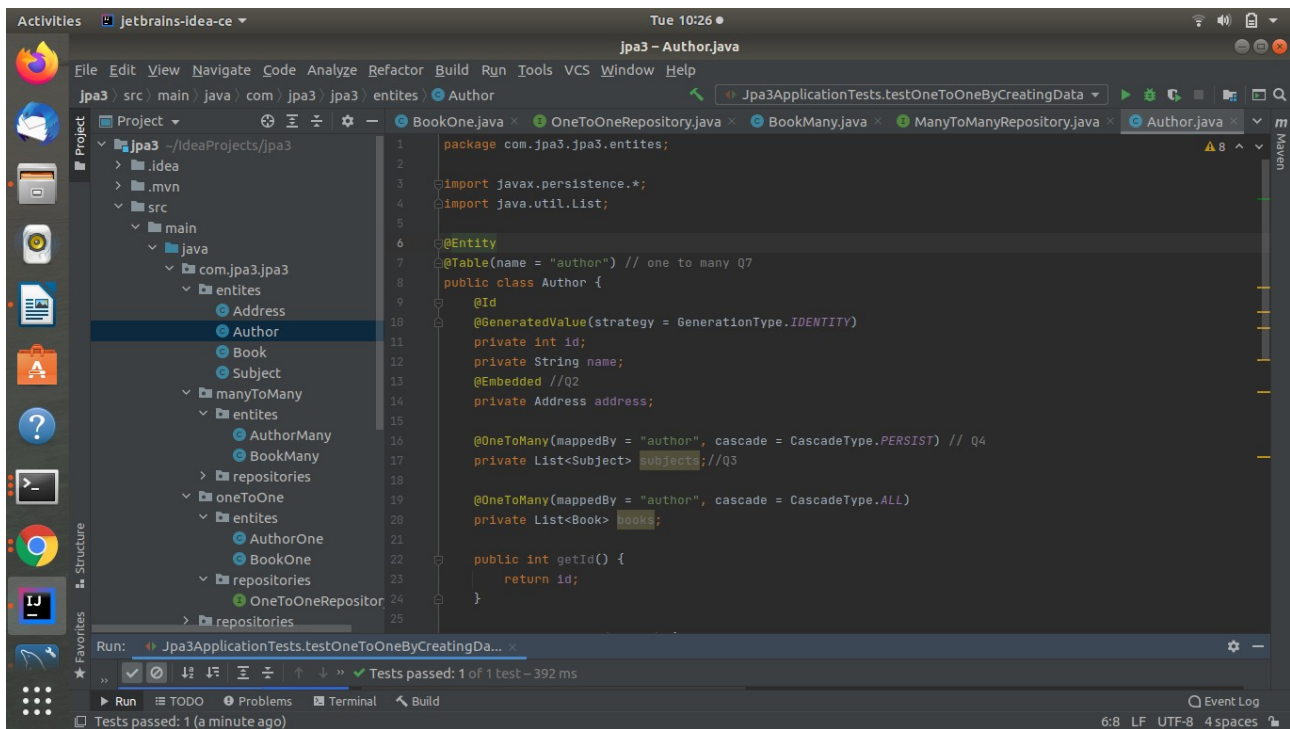


6. Implement One to One mapping between Author and Book.





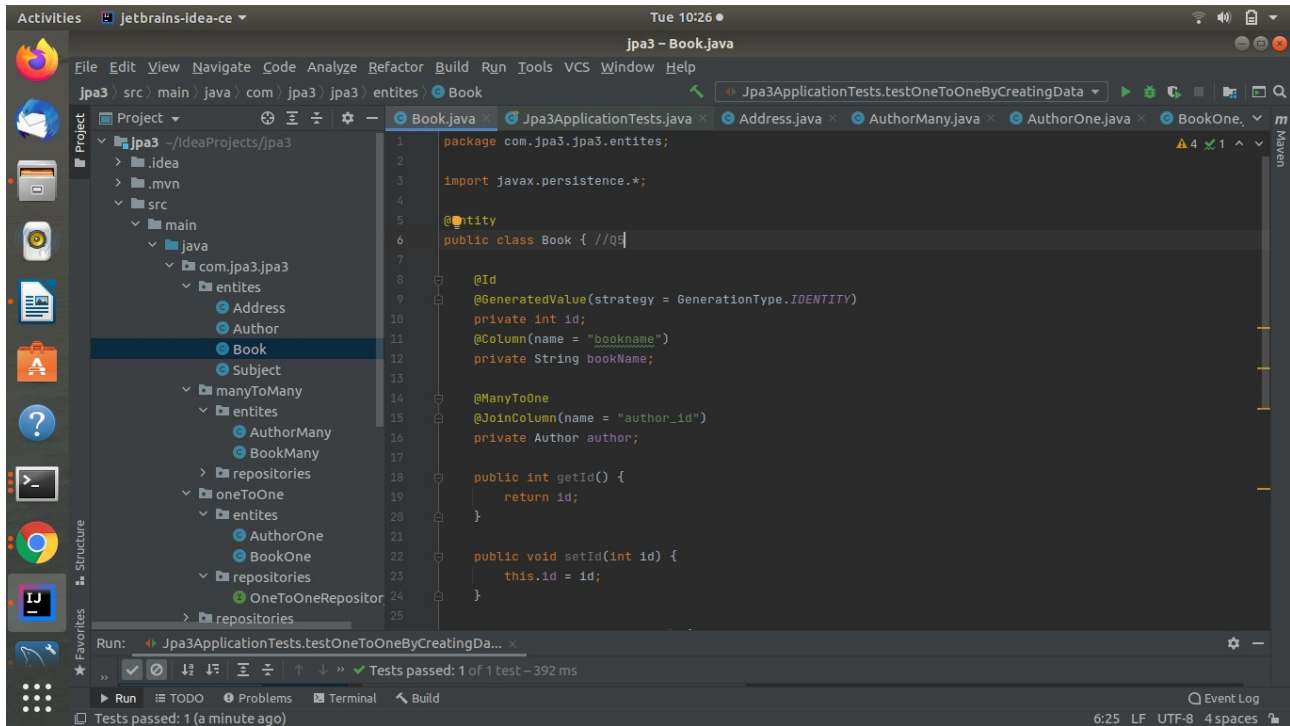
7. Implement One to Many Mapping between Author and Book(Unidirectional, BiDirectional and without additional table) and implement cascade save.



The screenshot shows the IntelliJ IDEA IDE with the `Author.java` file open. The project structure on the left indicates a package `com.jpaa3.jpaa3.entities` containing `Address`, `Author`, `Book`, and `Subject`. The `Author` class is annotated with `@Entity` and `@Table(name = "author")`. It has an `@Id` field `id` with `@GeneratedValue(strategy = GenerationType.IDENTITY)` and an `@Embedded` field `address`. It also has two `@OneToMany` relationships: `subjects` (mapped by `"author"`, cascade `PERSIST`) and `books` (mapped by `"author"`, cascade `ALL`). The `getId()` method returns `id`.

```
1 package com.jpaa3.jpaa3.entities;
2
3 import javax.persistence.*;
4 import java.util.List;
5
6 @Entity
7 @Table(name = "author") // one to many Q7
8 public class Author {
9     @Id
10    @GeneratedValue(strategy = GenerationType.IDENTITY)
11    private int id;
12    private String name;
13    @Embedded //Q2
14    private Address address;
15
16    @OneToMany(mappedBy = "author", cascade = CascadeType.PERSIST) // Q4
17    private List<Subject> subjects; //Q3
18
19    @OneToMany(mappedBy = "author", cascade = CascadeType.ALL)
20    private List<Book> books;
21
22    public int getId() {
23        return id;
24    }
25}
```

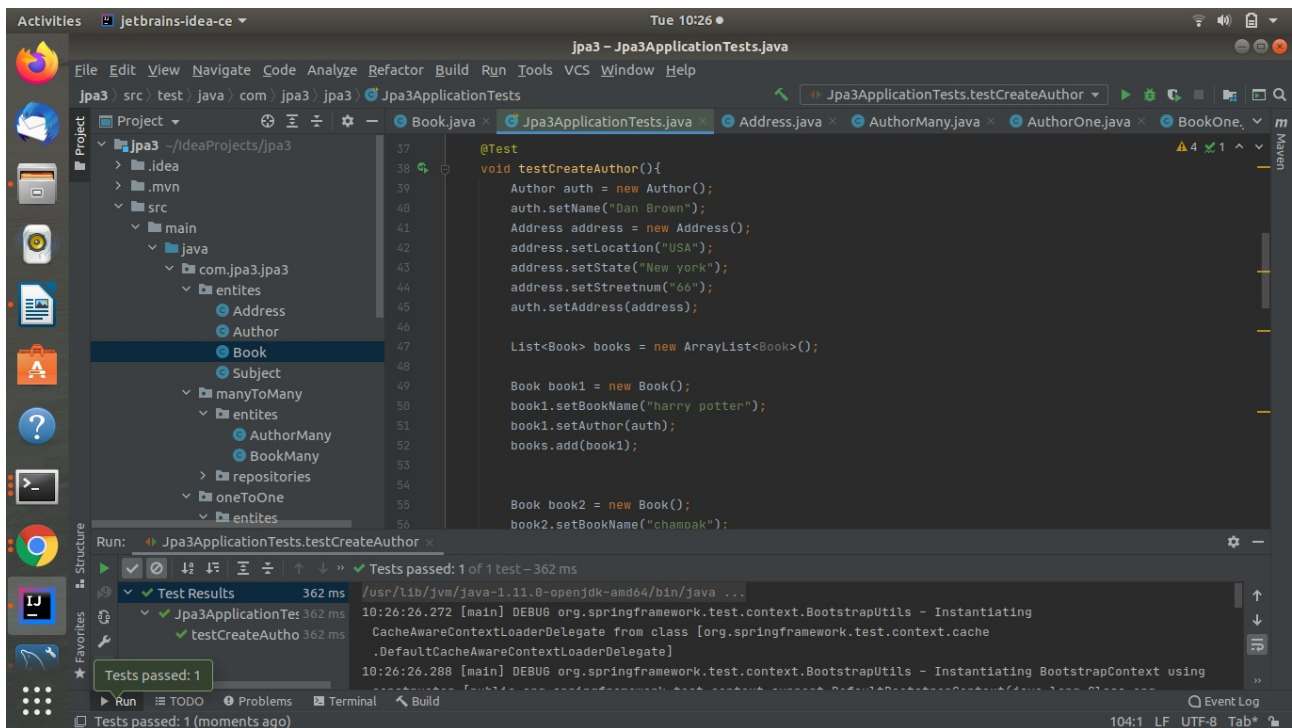
The bottom status bar shows "Tests passed: 1 of 1 test - 392 ms".



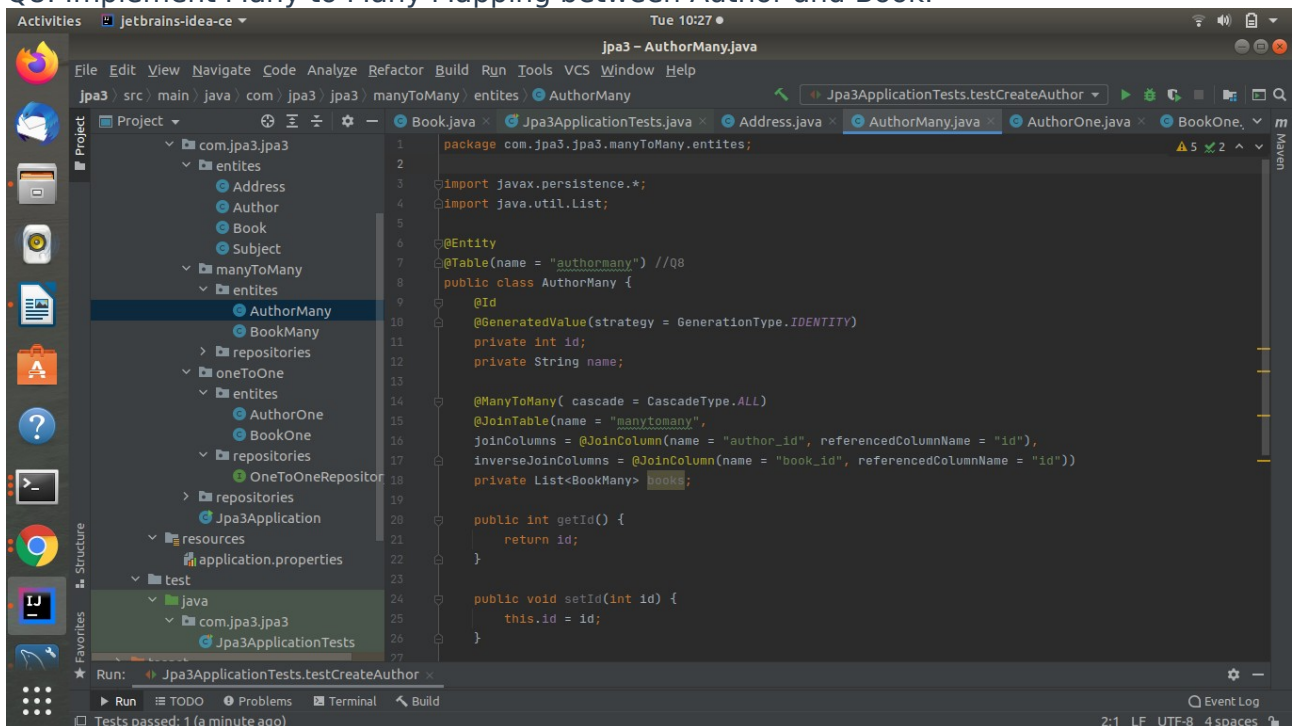
The screenshot shows the IntelliJ IDEA IDE with the `Book.java` file open. The project structure on the left is the same as the previous screenshot. The `Book` class is annotated with `@Entity` and has an `@Id` field `id` with `@GeneratedValue(strategy = GenerationType.IDENTITY)`. It has a `@Column` field `bookName` and a `@ManyToOne` relationship `author` (mapped by `"author_id"`). The `getId()` method returns `id`, and the `setId(int id)` method sets `this.id = id`.

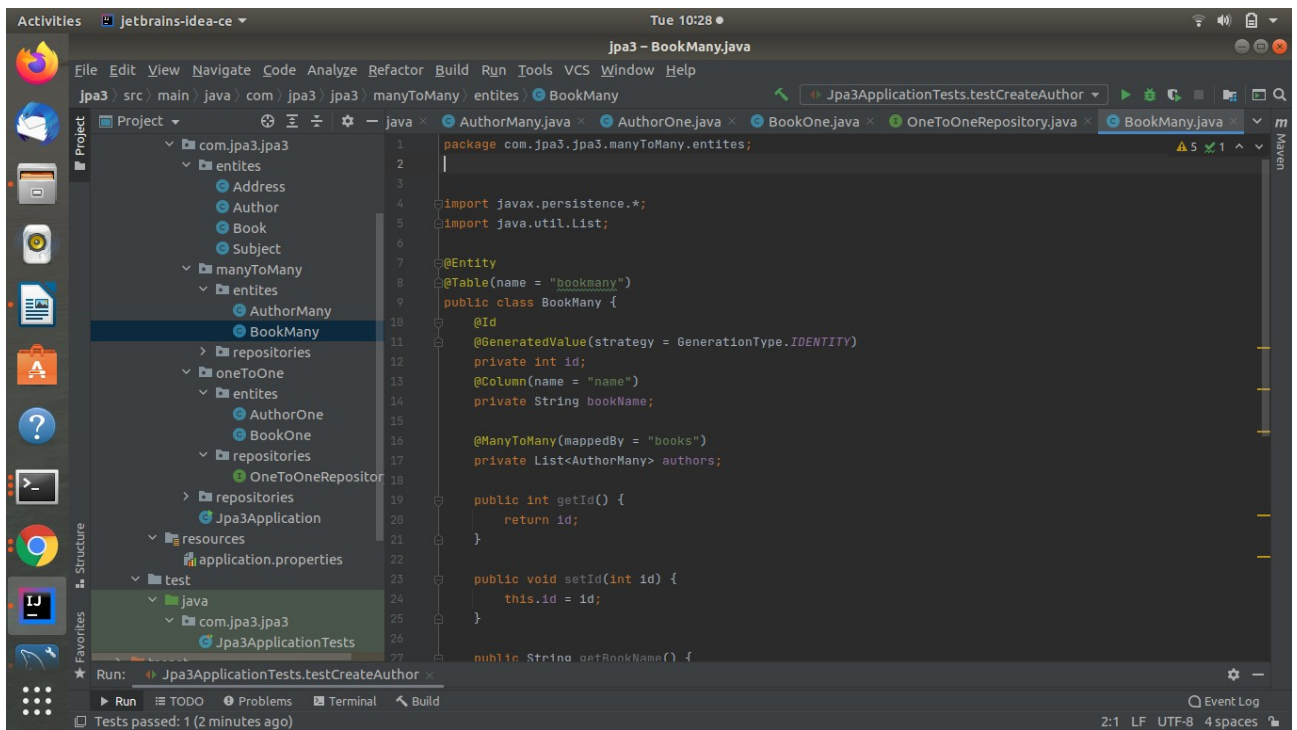
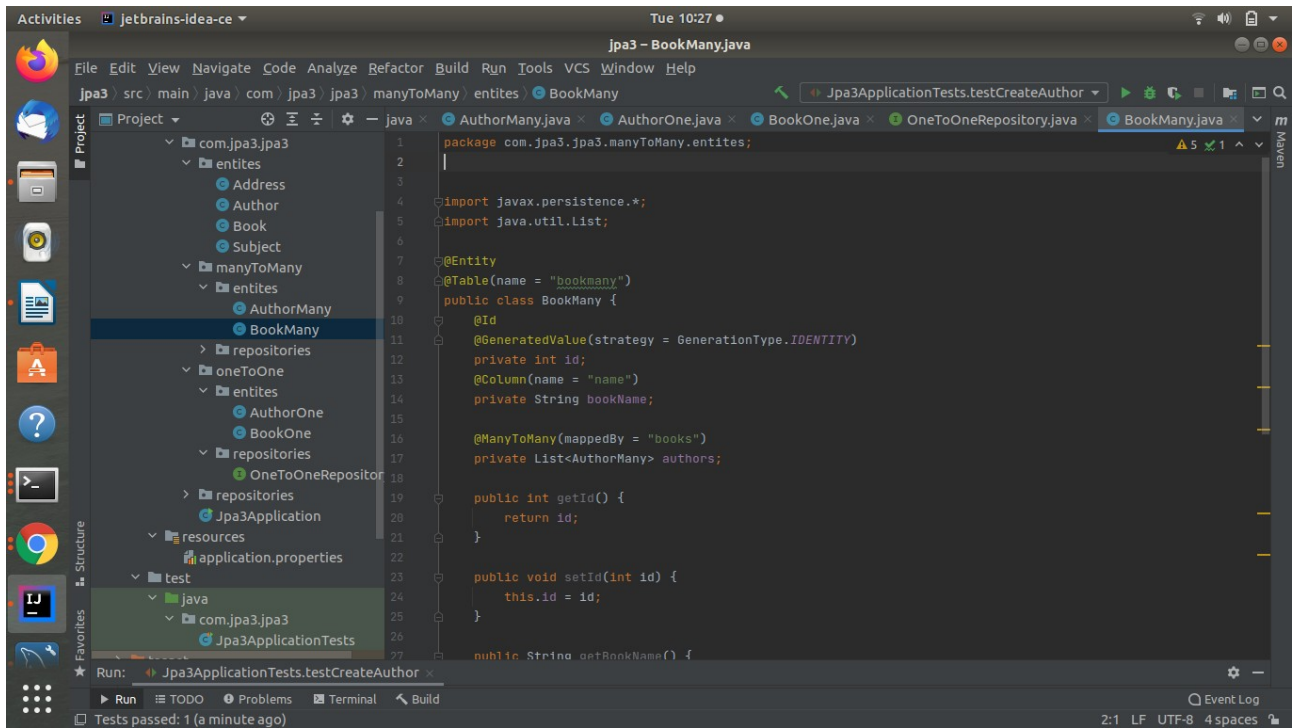
```
1 package com.jpaa3.jpaa3.entities;
2
3 import javax.persistence.*;
4
5 @Entity
6 public class Book { //Q5
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    private int id;
11    @Column(name = "bookname")
12    private String bookName;
13
14    @ManyToOne
15    @JoinColumn(name = "author_id")
16    private Author author;
17
18    public int getId() {
19        return id;
20    }
21
22    public void setId(int id) {
23        this.id = id;
24    }
25}
```

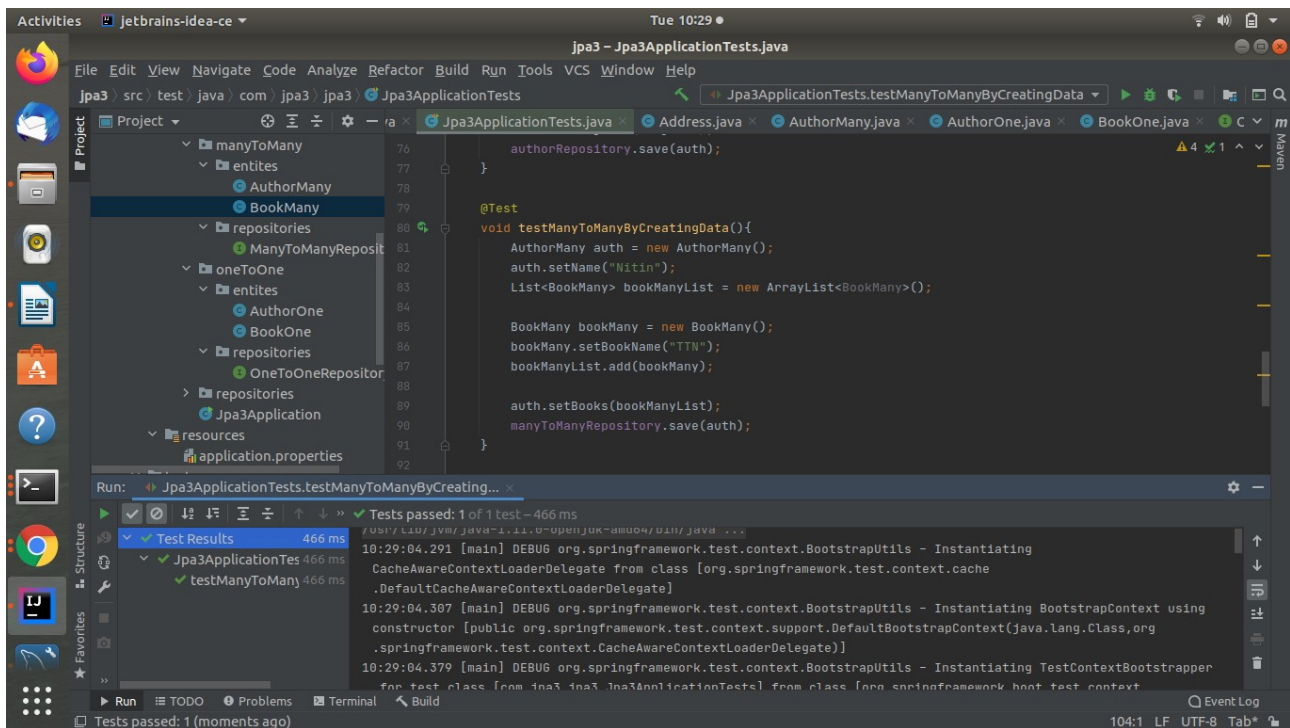
The bottom status bar shows "Tests passed: 1 of 1 test - 392 ms".



Q8: Implement Many to Many Mapping between Author and Book.







Q9: Which method on the session object can be used to remove an object from the cache?

Ans: 1) `evict()` is used to remove a particular object from cache associated with session.

2) You cannot disable the first level cache, it always enabled

3) Hibernate entities or database rows remain in cache only until Session is open, once Session is closed, all associated cached data is lost

Eg:

@Test

@transactional

```
public void testCaching(){
    Session session = new entityManager.unwrap(Session.class);
    Author author = new repository.findOne(1);
    repository.findOne(1);
    repository.findOne(1); // query will not be executed due to caching
    session.evict(author);
    repository.findOne(1); // query will be executed
}
```


Q10: What does @transactional annotation do?

Ans: The @Transactional annotation is metadata that specifies that an interface, class, or method must have transactional semantics; for example, "start a brand new read-only transaction when this method is invoked, suspending any existing transaction".

Eg:

```
@Transactional //whole transection won't commit due to transectional
public void testTransfer(int amount){
    Account account1 = new Account();
    Account account2 = new Account();
    account1.setBalance(account1.getBalance() - amount);
    if(true){
        throw new RuntimeException();
    }
    account2.setBalance(account2.getBalance() + amount);
}
```