Maksymilian Kempa 197259

Na implementację zadania składa się 5 plików:

1. Klasa Mage – deklaracja jednostki encyjnej, używanej do testowania. Zawiera pola *name*, *level*, gettery oraz settery.
2. Klasa MageRepository – implementacja repozytorium na obiekty klasy Mage. Obiekty są przechowywane w postaci kolekcji.
3. Klasa MageController – implementacja kontrolera do zarządzania obiektami typu Mage z repozytorium. Zawiera pole repository oraz 3 metody: find, delete oraz save.
4. EntityControllerTest – plik zawierający testy jednostkowe kontrolera z użyciem atrapy obiektu repozytorium (zaimplementowanej przy użyciu biblioteki Mockito).
5. EntityRepositoryTest - plik zawierający testy jednostkowe repozytorium.

MageRepository:

```
public Optional<Mage> find(String name) {
    return collection.stream()
        .filter(mage -> mage.getName().equals(name))
        .findFirst();
}
```

Metoda *find* służy do wyszukiwania maga w repozytorium po jego nazwie. Elementy z kolekcji są przetwarzane w sposób funkcyjny za pomocą strumienia danych. Najpierw filtrowane są one za pomocą funkcji filter, której argumentem jest wyrażenie lambda sprawdzające, czy nazwa obecnie przetwarzanego obiektu zgadza się z nazwą podaną jako parametr. Metoda *findFirst()* zwraca pierwszy znaleziony obiekt spełniający warunek filtrowania. Jeśli żaden obiekt nie pasuje do warunku, zwracany jest pusty obiekt typu Optional.

```
public void delete(String name) {
    if (this.find(name).isEmpty())
        throw new IllegalArgumentException("Mage doesn't exist.");
    else
        collection.removeIf(mage -> mage.getName().equals(name));
}
```

Metoda *delete* służy do usuwania maga z repozytorium. Najpierw wywołuje wyżej omówioną metodę *find* w celu stwierdzenia, czy mag o podanej nazwie rzeczywiście istnieje w bazie i może być usunięty. Jeśli nie, rzuca odpowiedni wyjątek, a w przeciwnym razie usuwa maga, którego nazwa odpowiada parametrowi naszej metody.

```java
public void save(Mage mage) {
    if (this.find(mage.getName()).isPresent())
        throw new IllegalArgumentException("Mage already exists.");
    else
        collection.add(mage);
}
```

Metoda *save* służy do zapisania maga do repozytorium i ma podobne działanie jak delete. Sprawdzane jest, czy mag istnieje już w bazie, jeśli tak, rzuca wyjątek, a jeśli nie, dodaje go do kolekcji.

MageController:

```java
public String find(String name) {
    Optional<Mage> mage = repository.find(name);
    if (mage.isPresent()) {
        return mage.get().toString();
    } else {
        return "not found";
    }
}
public String delete(String name) {
    try {
        repository.delete(name);
        return "done";
    } catch (IllegalArgumentException e) {
        return "not found";
    }
}
public String save(Mage mage) {
    try {
        repository.save(mage);
        return "done";
    } catch (IllegalArgumentException e) {
        return "bad request";
    }
}
```

Metody *find*, *delete* oraz *save* służą do obsługi wyników wywołania metod klasy MageRepository. Szczegółowa specyfikacja obiektów typu String, które mają zwracać poszczególne metody, znajduje się w instrukcji. W skrócie, kontroler służy do przyjmowania żądań od użytkownika, przekazywania ich do bazy danych, gdzie są wywoływane odpowiednie funkcje, a następnie na podstawie wyniku operacji przygotowanie danych do wyświetlenia użytkownikowi. Przykładowo, pomyślne wyszukiwanie maga za pomocą metody *find* spowoduje zwrócenie tekstowej reprezentacji wyszukiwanego obiektu, natomiast niepowodzenie - stringa „not found”.

EntityRepositoryTest:

```
@Mock
private MageRepository repository;

@BeforeEach
public void setup() {
    repository = new MageRepository(new ArrayList<>());
}
```

Na samym początku tworzona jest atrapa obiektu MageRepository. Obiekty Mock są w celu symulowania zachowania rzeczywistego obiektu w trakcie testowania. Znacznik @BeforeEach oznacza operacje wykonywane przed przeprowadzeniem każdego testu. W tym przypadku, przed każdym testem jest tworzona pusta atrapa repozytorium. Następnie w pliku są umieszczone testy, sprawdzające działanie wszystkich funkcjonalności repozytorium.

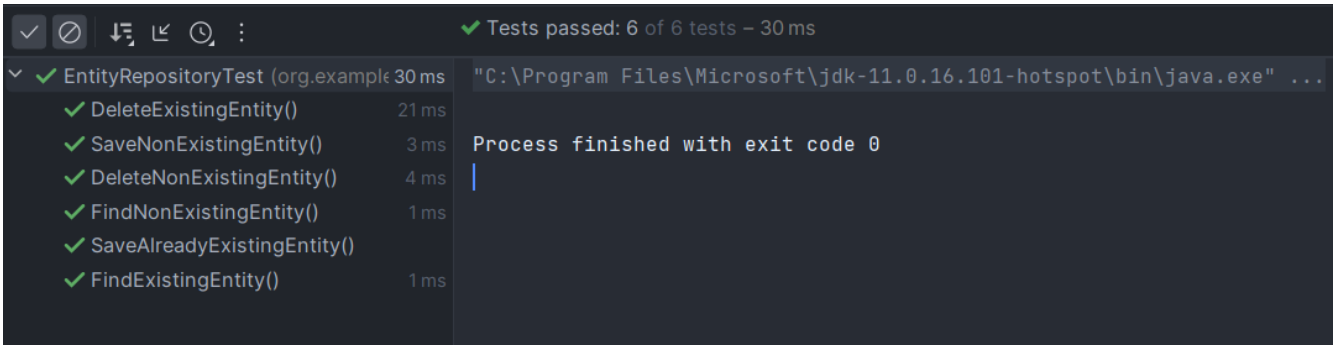
```
@Test
public void SaveNonExistingEntity() {
    Mage mage = new Mage("Harry Potter", 100);
    repository.save(mage);

    Optional<Mage> result = repository.find("Harry Potter");
    assertTrue(result.isPresent());
    assertEquals(mage, result.get());
}
```

Przykładowo, powyższy test sprawdza zachowanie programu po próbie wstawienia do repozytorium nowego obiektu o nazwie, której nie ma jeszcze żaden obiekt znajdujący się w naszej kolekcji. Operacja powinna zakończyć się sukcesem – na początku tworzona jest pusta atrapa, do której przypisujemy nowego maga, w związku z czym nie powinno być żadnych konfliktów.

```
@Test
public void DeleteNonExistingEntity() {
    assertThrows(IllegalArgumentException.class,()->repository.delete("Ron Weasley"));
}
```

Powyższy test z kolei testuje usunięcie nieistniejącego obiektu z repozytorium. Zgodnie z instrukcją test zakończy się sukcesem, jeśli metoda rzuci oczekiwany wyjątek.

Jak widać, wszystkie 6 testów zamieszczonych w pliku kończy się powodzeniem:



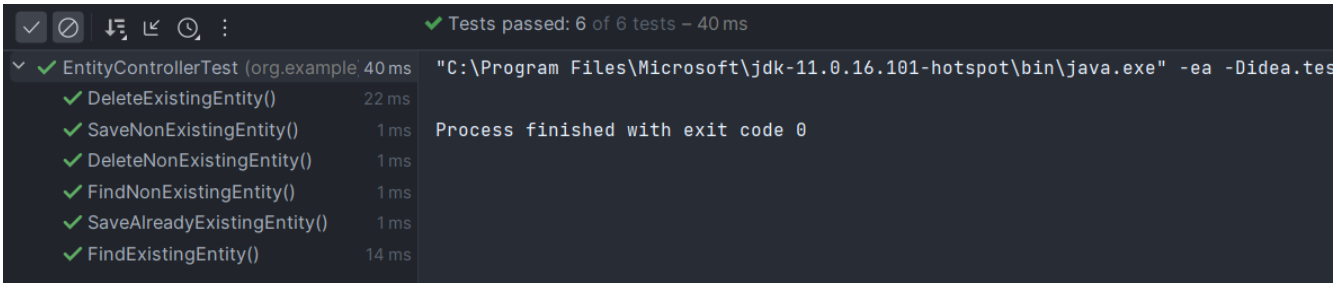
EntityControllerTest:

```
@Mock
private MageRepository repository;

private MageController controller;

@BeforeEach
public void setup() {
    repository = new MageRepository(new ArrayList<>());
    controller = new MageController(repository);
}
```

Test kontrolera jest bardzo podobny do testu repozytorium. Inicjalizacja różni się jedynie tym, że oprócz kontrolera musimy stworzyć jeszcze atrapę repozytorium, które jest parametrem obiektu klasy MageController. Po tych linijkach kodu znajdują się testy jednostkowe.

```
@Test
public void SaveNonExistingEntity() {
    String result = controller.save(new Mage("Harry Potter", 100));
    assertEquals(result, "done");
}
```

Przykładowo, powyższy test sprawdza poprawność wykonania metody *save* kontrolera. *AssertEquals* to asercja, która sprawdza, czy wynikiem operacji jest String „done”. Powinien być, gdyż próbujemy zapisać nieistniejącego jeszcze w bazie danych maga. Jeżeli zwróci coś innego, test zakończy się niepowodzeniem.



Również w tym przypadku wszystkie testy zakończyły się powodzeniem – program za każdym razem zwrócił oczekiwany łańcuch znaków.