CS 5200

Milestone 4

Team 6

Screenshot 1:

- Used the Create, read, update and delete operations to create a new Person named “nathan” updated this Person to “nathanUpdated”. Finally deleted this Person.

```
// INSERT objects from our model.
Person person1 = new Person( username: "nathan");
person1 = personsDao.create(person1);

// READ.
Person p1 = personsDao.getPersonByUsername("nathan");
System.out.format("Created and Reading person: %s\n", p1.getUsername());

// Update
Person p3 = personsDao.updateUsername(p1, newUsername: "nathanUpdated");
System.out.format("Updating Person: name:%s \n",
  p3.getUsername());

// Delete.
System.out.println("Successfully deleted Person nathanUpdated.");
personsDao.delete(p3);

// Spacing
System.out.println();
```

-

```
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
Created and Reading person: nathan
Updating Person: name:nathanUpdated
Successfully deleted Person nathanUpdated.
```

-

Screenshot 2:

- Used the Create, read, update and delete operations to create a new administrator "elsaTa" updated this Administrator to "panXu1" and finally deleted this Administrator.
-

```
// INSERT objects from our model.
Administrator admin1 = new Administrator( username: "elsaTa");
admin1 = administratorsDao.create(admin1);

// READ.
Administrator a1 = administratorsDao.getAdministratorFromUserName("elsaTa");
System.out.format("Created and reading administrator: %s \n",
  a1.getUsername());

// Update
Administrator a3 = administratorsDao.updateUserName(a1, newUserName: "panXu1");
System.out.format("Updating administrators from name elsaTa to :%s \n",
  a3.getUsername());

// Delete.
System.out.println("Successfully deleted the administrator panXu1");
administratorsDao.delete(a3);

// Spacing
System.out.println();
```

-

```
Created and reading administrator: elsaTa
Updating administrators from name elsaTa to :panXu1
Successfully deleted the administrator panXu1
```

-

Screenshot 3:

Created 2 users “nathanY56” and “deathBySora” and then used the .getUsersFromUserName() method to find these 2 users. Further used the Update operation to update “nathanY56” to “newNathanY56”. Finally deleted this user.

```
// INSERT objects from our model.
User user1 = new User( username: "nathanY56");
user1 = usersDao.create(user1);

User user2 = new User( username: "deathBySora");
user2 = usersDao.create(user2);

// READ.
User u1 = usersDao.getUserFromUserName("nathanY56");
System.out.format("Successfully created and found user1: u:%s using the .getUserFromUserName() method\n",
  u1.getUsername());

User u2 = usersDao.getUserFromUserName("deathBySora");
System.out.format("Successfully created and found user2: u:%s using the .getUserFromUserName() method \n",
  u2.getUsername());

// Update.
User u3 = usersDao.updateUserName(u1,  newUserName: "newNathanY56");
System.out.format("Updating users: from nathanY56 to new name:%s \n",
  u3.getUsername());

// Delete.
System.out.format("Successfully deleted deathbysora \n");
usersDao.delete(usersDao.getUserFromUserName("deathbysora"));
System.out.format("Successfully deleted " + u2.getUsername() + "\n");
usersDao.delete(u3);

// Spacing
System.out.println();
```

```
Successfully created and found user1: u:nathanY56 using the .getUserFromUserName() method
Successfully created and found user2: u:deathBySora using the .getUserFromUserName() method
Updating users: from nathanY56 to new name:newNathanY56
Successfully deleted deathbysora
Successfully deleted deathBySora
```

Screenshot 4:

Again played around and used the CRUD operations. We created 2 beers “Belgian Ale” and “Larger” and then read them. Then we updated the “Belgian Ale” into “IPA”. Also updated the “Pizza” we created into “Fries”. Then we deleted everything.

```
// Insert BeerStyle & Food
BeerStyle beerStyle1 = new BeerStyle("Belgian Ale");
beerStyle1 = beerStyleDao.create(beerStyle1);
BeerStyle beerStyle2 = new BeerStyle("Larger");
beerStyle2 = beerStyleDao.create(beerStyle2);

Food food1 = new Food( foodName: "Pasta", beerStyle1);
food1 = foodDao.create(food1);
Food food2 = new Food( foodName: "Pizza", beerStyle2);
food2 = foodDao.create(food2);

// Read BeerStyles & Food
BeerStyle bs1 = beerStyleDao.getBeerStyle("Belgian Ale");
System.out.format("Successfully created and reading beerStyle: s:%s \n",
  bs1.getStyle());

BeerStyle bs2 = beerStyleDao.getBeerStyle("Larger");
System.out.format("Successfully created and reading beerStyle: s:%s \n",
  bs2.getStyle());
```

```
  // Update BeerStyle & food
  BeerStyle bs3 = beerStyleDao.updateStyle(beerStyle1,  newStyle: "IPA");
  System.out.format("Updating beerStyle1 to new beerStyle3: s:%s \n",
    bs3.getStyle());

  Food f1 = foodDao.updateFoodName(food2,  newFoodName: "Fries");
  System.out.format("Updating food: from pizza to f:%s \n",
    f1.getFoodName());

  beerStyleDao.delete(beerStyleDao.getBeerStyle("larger"));
  beerStyleDao.delete(beerStyleDao.getBeerStyle("ipa"));
  // Delete BeerStyle & Food
  foodDao.getFoodByName( foodName: "fries").forEach((f) -> foodDao.delete(f));
  foodDao.getFoodByName( foodName: "pasta").forEach((f) -> foodDao.delete(f));
  foodDao.getFoodByName( foodName: "pizza").forEach((f) -> foodDao.delete(f));
```

```
Successfully created and reading beerStyle: s:Belgian Ale
Successfully created and reading beerStyle: s:Larger
Updating beerStyle1 to new beerStyle3: s:IPA
Updating food: from pizza to f:Fries
```

Advanced Feature Screenshot:

For our advanced feature, we created a method "mockFindSimilarBeers()" that takes in a beer name and then based on its attribute scores such as appearances, aroma, and palate, and goes through the database to find beers that have similar attribute scores. We have adjusted our advanced feature slightly from where we initially wrote in PM1. This still works along the lines of helping users find new beers that they can discover based on their preferences.

This is the main method that runs:

```
1 usage  deathbysora *
private static void mockFindSimilarBeers() {
    BeersDao beersDao = BeersDao.getInstance();

    Beer beer = new Beer( id: 2,  name: "Abeta Amber Lager", Float.valueOf( f: 5),  brewerId: 1, new BeerStyle("Vienna"));
    System.out.println("Find similarly rated beers for: " + beer.getName());
    beersDao.getSimilarBeers(beer).forEach((b) -> System.out.println("* " + b.getName()));
}
```

Notice it will call the .getSimilarBeers():

```
/**
 * User can get a list of similar beer recommendations based on the beer
 * they selected.
 *
 * @param beer the beer object that user selected
 * @return a list of beers (could be just one or null) which the metrics are similar to the
 * recommended beer.
 */
1 usage  deathbysora *
public List<Beer> getSimilarBeers(Beer beer) {
    BeerReviewsDao beerReviewsDao = BeerReviewsDao.getInstance();
    BeerReview averageReview = beerReviewsDao.findAverageReviewOf(beer);
    List<BeerReview> reviews = beerReviewsDao.findSimilarReviews(averageReview,  limit: 5);
    return reviews.stream().map((r) -> getBeerById(r.getId())).collect(Collectors.toList());
```

Please see that our getSimilarBeers() method would call additional methods that you would be able to trace under the BeerStylesDao.java file.

For this example, we used "Abeta Amber Lager" as an example and our tool will provide the top 5 similarly rated beers. Please see screenshot below that give s us the top 5 similar results.

```
"C:\Program Files\Java\jdk1.8.0_202\bin\java.exe" ...
Find similarly rated beers for: Abeta Amber Lager
* La Saint-Pierre Blonde de l`Oncle Hansi
* Abita Amber Lager
* Abita Andygator
* Abita Bock
* Abita Fall Fest

Process finished with exit code 0
```