**Создаем объект**

И так, чтобы что-то тестировать, нам для начала нужен объект тестирования. Перед нами стоит задача.

1. Нам нужен объект, который будет хранить информацию о Пользователе.  
   1. Id — нужно считать по порядку добавления нового пользователя.
   2. Имя пользователя.
   3. Его возраст.
   4. Пол (male/female)
2. Нужно предусмотреть хранение списка пользователей.
3. Класс должен уметь.
   * **Формировать список всех пользователей.**
   * **Формировать список пользователей по полу (MALE/FEMALE).**
   * **Возвращать количество пользователей в общем списке, и посчитать количество по признаку пола пользователя.**
   * **Посчитать общую сумму по возрасту пользователей, так же учесть по признаку пола.**
   * **Посчитать средний возраст, как общий так и по признаку пола.**

**Создадим Java класс User** который будет создавать/проверять/правильно выводить переменные:

package example2;   
  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Objects;  
  
public class User {  
 private String sex;  
 private int age;  
 private int id;  
 private String name;  
 private static Map<Integer, User> *allUsers* = new HashMap<>();  
 private static int *countId* = 0;  
  
 public User(String name, int age, String sex) {  
 //сразу делаем недопустимым ввод однозначно неправильных значений при создании объекта

// класса + эти же ограничения необходимо прописать и в set-ах  
 if (name != null && !name.isEmpty() && age > 0 && sex != null) {  
 this.name = name;  
 this.age = age;  
 this.sex = sex;  
  
 //проверяем наличие такого объекта в allUsers, если нет, то  
 // увеличиваем счетчик на 1 и добавляем объект в список  
 if (!hasUser()) {  
 *countId*++;  
 this.id = *countId*;  
 *allUsers*.put(id, this);//this здесь это объект этого-же класса  
 }  
 }  
 }  
 public User() {}  
  
 public int getId() { return id; }  
 public int getAge() { return age; }  
 public String getName() { return name; }  
 public String getSex() { return sex; }  
 public static Map<Integer, User> getAllUsers() { return *allUsers*; }  
 public void setSex(String sex) { this.sex = sex; }  
 public void setAge(int age) { this.age = age; }  
 public void setId(int id) { this.id = id; }  
 public void setName(String name) { this.name = name; }  
 public static void setAllUsers(Map<Integer, User> allUsers) { User.*allUsers* = allUsers; }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 User user = (User) o;  
 return Objects.*equals*(name, user.name) &&  
 age == user.age && sex == user.sex;  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(age, name, sex);  
 }  
  
 @Override  
 public java.lang.String toString() {  
 return "example2.User{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", age=" + age +  
 ", sex=" + sex +  
 '}';  
 }  
  
 private boolean hasUser(){  
 for (User user: *allUsers*.values()){  
 if (user.equals(this) && user.hashCode() == this.hashCode()) {  
 return true;  
 }  
 }  
 return false;  
 }  
}

**Создадим Java класс Sex** с константами:

package example2;  
  
public class Sex {  
 public static final String *MALE* = "male";  
 public static final String *FEMALE* = "female";  
}

**Создадим Java класс Logics** с логикой расчетов, использующий переменные из класса User:

package example2;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Logics {  
  
 //Формировать список всех пользователей.  
 public static List<User> **listAllUsers()**{  
 return new ArrayList<>(User.*getAllUsers*().values());  
 }  
 //Формировать список пользователей по полу(MALE/FEMALE).  
 public static List<User> **listAllUsers(String sex)**{  
 List<User> lAllUsers = new ArrayList<>();  
 for (User user : User.*getAllUsers*().values()){  
 if (user.getSex() == sex){  
 lAllUsers.add(user);  
 }  
 }  
 return lAllUsers;  
 }  
  
 //Возвращать количество пользователей в общем списке  
 public static int **getHowManyUsers**(){  
 return *listAllUsers*().size();  
 }  
 //Возвращать количество пользователей в общем списке по признаку пола  
 public static int **getHowManyUsers**(String sex){  
 return *listAllUsers*(sex).size();  
 }  
  
 //Посчитать общую сумму по возрасту пользователей  
 public static int **getAllAgeUsers()**{  
 int countAge = 0;  
 for (User user : *listAllUsers*()){  
 countAge += user.getAge();  
 }  
 return countAge;  
 }  
 //Посчитать общую сумму по возрасту пользователей с учетом признака пола  
 public static int **getAllAgeUsers(String sex)**{  
 int countAge = 0;  
 for (User user : *listAllUsers*(sex)){  
 countAge += user.getAge();  
 }  
 return countAge;  
 }  
  
 //Посчитать общий средний возраст  
 public static int **getAverageAgeOfAllUsers()**{  
 return *getAllAgeUsers*() / *getHowManyUsers*();  
 }  
 //Посчитать средний возраст по признаку пола  
 public static int **getAverageAgeOfAllUsers(String sex)**{  
 return *getAllAgeUsers*(sex) / *getHowManyUsers*(sex);  
 }  
}

**Вот так теперь будет выглядеть наш класс тестов с аннотацией @Before Each:**

package example2;  
  
import org.junit.Assert;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import java.time.Duration;  
import static java.time.Duration.*ofSeconds*;  
import static org.junit.jupiter.api.Assertions.*assertTimeout*;  
import static org.junit.jupiter.api.Assertions.*assertTimeoutPreemptively*;  
  
class LogicsTest {  
  
 private User user;  
 private User user1;  
 private User user2;  
  
 @BeforeEach  
 void setUp() throws Exception {  
 user = new User("Евгений", 35, Sex.*MALE*);  
 user1 = new User("Марина", 34, Sex.*FEMALE*);  
 user2 = new User("Алина", 7, Sex.*FEMALE*);  
 }  
  
 /\*Если нужно остановить тест через необходимое время, то  
 @Test  
 void getAllUsers() {  
 //нужно писать assertTimeoutPreemptively() и тогда выполнение метода  
 //принудительно остановится через необходимое время и будет написан наш текст  
 assertTimeoutPreemptively(Duration.ofMillis(3), () -> {  
  
 //если написать через assertTimeout(), то метод будет выполняться все  
 // необходимое ему время, а потом напишет наш текст  
 assertTimeout(ofSeconds(1), () -> {  
  
 while (true); //или любой текст метода  
 },() -> "Тест выполняется больше 3 ms");  
 }

\*/

---------------------------------------------------------------------------------------------

//Следующие четыре теста будут проверять на правильность введения *(написанную  
 // в конструкторе строку)* для начальных параметров. При этом они будут ловить только  
 // если в классе User удалить или закоментировать строку  
 // "if (name != null && !name.isEmpty() && age > 0 && sex != null) {"  
 //А при этой строке предварительного отсева, при попытке неправильного ввода  
 // параметров, покажут ошибку проверки других методов.

@*Test*  
 public void newUser\_EMPTY\_NAME() {  
 for (User user : Logics.*listAllUsers*()){  
 if (user.getName() != null && user.getName().isEmpty()) {  
 Assert.*fail*("Попытка создания пользователя с пустым именем");  
 }}}

@*Test*  
 public void newUser\_NAME\_ZERO() {  
 for (User user : Logics.*listAllUsers*()){  
 if (user.getName() == null || user.getName().isEmpty()) {  
 Assert.*fail*("Попытка создания пользователя с пустым именем");  
 }}}

@*Test*  
 public void newUser\_AGE\_ZERO() {  
 for (User user : Logics.*listAllUsers*()) {  
 if (user.getAge() <= 0) {  
 Assert.*fail*("Попытка создания пользователя c не допустимым возрастом");  
 }}}

@*Test*  
 public void newUser\_SEX\_NO\_NULL() {  
 for (User user : Logics.*listAllUsers*()) {  
 if (user.getSex() == null) {  
 Assert.*fail*("Попытка создания пользователя с указанием пола = null");  
 }}}  
  
---------------------------------------------------------------------------------------------  
  
 @*Test*  
 void **listAllUsers**() {  
 List<User> actual = Logics.*listAllUsers*();  
  
 List<User> expected = new ArrayList<>();  
 expected.add(user);  
 expected.add(user1);  
 expected.add(user2);  
  
 Assert.*assertEquals*(expected, actual);  
 }  
  
 @*Test*  
 void **listAllUsers**\_NO\_NULL() {  
 List<User> actual = Logics.*listAllUsers*();  
 Assert.*assertNotNull*(actual);  
 }  
  
 @*Test*  
 void **listAllUsers**\_MALE() {  
 List<User> expected = Logics.*listAllUsers*(Sex.*MALE*);  
  
 List<User> actual = new ArrayList<>();  
 actual.add(user);  
  
 Assert.*assertEquals*(expected, actual);  
 }  
  
 @*Test*  
 void **listAllUsers**\_MALE\_NO\_NULL() {  
 //добавим проверку на null  
 List<User> actual = Logics.*listAllUsers*(Sex.*MALE*);  
 Assert.*assertNotNull*(actual);  
 }  
  
 @*Test*  
 void **listAllUsers**\_FEMALE() {  
 List<User> actual = Logics.*listAllUsers*(Sex.*FEMALE*);  
  
 List<User> expected = new ArrayList<>();  
 expected.add(user1);  
 expected.add(user2);  
  
 Assert.*assertEquals*(expected, actual);  
 }

@*Test*  
 void **getAllUsers**\_FEMALE\_NO\_NULL() {  
 //добавим проверку на null  
 List<User> actual = Logics.*listAllUsers*(Sex.*FEMALE*);  
 Assert.*assertNotNull*(actual);  
 }

---------------------------------------------------------------------------------------------  
  
 @*Test*  
 void **getHowManyUsers**() {  
 int actual = Logics.*getHowManyUsers*();  
 int expected = 3;  
 Assert.*assertEquals*(expected, actual);  
 }  
  
 @*Test*  
 void **getHowManyUsers**\_MALE() {  
 int actual = Logics.*getHowManyUsers*(Sex.*MALE*);  
 int expected = 1;  
 Assert.*assertEquals*(expected, actual);  
 }  
  
 @*Test*  
 void **getHowManyUsers**\_FEMALE() {  
 int actual = Logics.*getHowManyUsers*(Sex.*FEMALE*);  
 int expected = 2;  
 Assert.*assertEquals*(expected, actual);  
 }

---------------------------------------------------------------------------------------------  
  
 @*Test*  
 void **getAllAgeUsers**() {  
 int actual = Logics.*getAllAgeUsers*();  
 int expected = 35 + 34 + 7;  
 Assert.*assertEquals*(expected, actual);  
 }  
  
 @*Test*  
 void **getAllAgeUsers**\_MALE() {  
 int actual = Logics.*getAllAgeUsers*(Sex.*MALE*);  
 int expected = 35;  
 Assert.*assertEquals*(expected, actual);  
 }  
  
 @*Test*  
 void **getAllAgeUsers**\_FEMALE() {  
 int actual = Logics.*getAllAgeUsers*(Sex.*FEMALE*);  
 int expected = 34 + 7;  
 Assert.*assertEquals*(expected, actual);  
 }  
}

Вот список аннотаций для JUnit с ними однозначно жить проще.

@Test – определяет что метод method() является тестовым.

@Before – указывает на то, что метод будет выполнятся перед каждым тестируемым методом @Test.

@After – указывает на то что метод будет выполнятся после каждого тестируемого метода @Test

@BeforeClass – указывает на то, что метод будет выполнятся в начале всех тестов,

а точней в момент запуска тестов(перед всеми тестами @Test).

@AfterClass – указывает на то, что метод будет выполнятся после всех тестов.

@Ignore – говорит, что метод будет проигнорирован в момент проведения тестирования.

(expected = Exception.**class**) – указывает на то, что в данном тестовом методе

вы преднамеренно ожидаете Exception.

(timeout = 100) – указывает, что тестируемый метод не должен занимать больше чем 100 миллисекунд.

Основные методы класса Assert для проверки:

***fail***(String) – указывает на то что бы тестовый метод завалился при этом выводя текстовое сообщение.

***assertTrue***([message], **boolean** condition) – проверяет, что логическое условие истинно.

***assertsEquals***([String message], expected, actual) – проверяет, что два значения совпадают.

Примечание: для массивов проверяются ссылки, а не содержание массивов.

***assertNull***([message], object) – проверяет, что объект является пустым null.

***assertNotNull***([message], object) – проверяет, что объект не является пустым null.

***assertSame***([String], expected, actual) – проверяет, что обе переменные относятся к одному объекту.

***assertNotSame***([String], expected, actual) – проверяет, что обе переменные относятся к разным объектам.

Вот так мы можем добавить зависимость JUnit 4.12 в Maven

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>