Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

**Лабораторная работа №2**

**Дисциплина «Тестирование программного обеспечения»**

**Вариант 19**

**Выполнил:**

Съестов Дмитрий Вячеславович

Группа P3317

**Преподаватель:**

Исаев Илья Владимирович

Санкт-Петербург

2019

**Задание**

Разработка консольного приложения “Музыкальный каталог”.

При запуске программы пользователю предлагается справочная информация по использованию. Далее предлагается ввести команду, позволяющую выполнить одно из действий:

* осуществить поиск музыкальной композиции в каталоге по определенному критерию;
* вывести информацию о всех существующих в каталоге композициях;
* добавить информацию о композиции в каталог;
* удалить существующую в каталоге запись;
* выйти из программы.

Критериями поиска могут служить: имя (название) автора/исполнителя или название композиции. В качестве результата поиска в консоль должен выводиться список композиций в виде «исполнитель – название». Удаление или добавление записи осуществляется после ввода всей информации о композиции.

**Функциональные требования**

|  |  |  |
| --- | --- | --- |
| **ID** | **Название** | **Описание** |
| FR0 | Консольный вывод | Взаимодействие с пользователем должно осуществляться с помощью консольного интерфейса. |
| FR1 | Консольный ввод | Система должна поддерживать консольный ввод. |
| FR2 | Название страницы | Система должна явно идентифицировать текущую страницу, выводя её название в консоль. |
| FR3 | Регистрация | Система должна позволять пользователю создать учёную запись. |
| FR4 | Вход | Система должна позволять пользователю войти в существующую учётную запись. |
| FR5 | Минимальная длина пароля | Система должна проверять, что длина пароля не меньше 6 символов. |
| FR6 | Проверка надёжности пароля | Система должна проверять, что пароль содержит как буквы, так и цифры. |
| FR7 | Хранение паролей | Пользовательские пароли должны храниться в виде хэш-кода. |
| FR8 | Проверка паролей при входе | Система должна проверять пароль и запрещать вход при его несоответствии. |
| FR9 | Использование базы данных | Система должна хранить все данные в БД на удалённом сервере. |
| FR10 | Главное меню | Система должна выводить список всех возможных действий в главном меню. |
| FR11 | Выбор пункта меню | Выбор пункта меню должен осуществляться путём ввода его номера. |
| FR12 | Вывод всех записей | Система должна позволять пользователю вывести список всех существующих записей. |
| FR13 | Подробная информация о записи | Система должна позволять пользователю просматривать подробную информацию о записи. |
| FR14 | Удаление | Система должна позволять пользователю удалять композиции. |
| FR15 | Добавление | Система должна позволять пользователю добавлять новые композиции. |
| FR16 | Поиск по названию | Система должна позволять пользователю искать композиции по названию. |
| FR17 | Поиск по исполнителю | Система должна позволять пользователю искать композиции по исполнителю. |
| FR18 | Поиск по году | Система должна позволять пользователю искать композиции по году выпуска. |
| FR19 | Регистронезависимый поиск | Поиск должен быть регистронезависимым. |
| FR20 | О программе | Система должна позволять пользователю просматривать информацию о программе. |
| FR21 | Выход из программы | Система должна позволять пользователю выйти из программы. |

**DAO-классы**

Для работы с БД я использовал ORM-фреймворк Exposed.

Пользователь:

package dmitry.tpo2.entity  
  
import org.jetbrains.exposed.dao.\*  
  
object Users : IntIdTable() {  
 val username= varchar("username", 20).uniqueIndex()  
 val hashedPassword = integer("hashed\_password")  
}  
  
class User(id: EntityID<Int>) : IntEntity(id) {  
 companion object : IntEntityClass<User>(Users)  
  
 var username by Users.username  
 var hashedPassword by Users.hashedPassword  
}

Композиция:

package dmitry.tpo2.entity  
-  
import org.jetbrains.exposed.dao.\*  
  
object MusicTracks : IntIdTable() {  
 val name = varchar("name", 20)  
 val artist = reference("artistID", Artists).nullable()  
 val year = integer("year").check **{** it.greater(0) **}**.nullable()  
}  
  
class MusicTrack(id: EntityID<Int>) : IntEntity(id) {  
 companion object : IntEntityClass<MusicTrack>(MusicTracks)  
  
 var name by MusicTracks.name  
 var artist by Artist optionalReferencedOn MusicTracks.artist  
 var year by MusicTracks.year  
  
 override fun toString(): String {  
 return "#$id\t${artist ?: "Неизвестный исполнитель"} - $name ($year)"  
 }  
}

Исполнитель:

package dmitry.tpo2.entity  
  
import org.jetbrains.exposed.dao.EntityID  
import org.jetbrains.exposed.dao.IntEntity  
import org.jetbrains.exposed.dao.IntEntityClass  
import org.jetbrains.exposed.dao.IntIdTable  
import org.jetbrains.exposed.sql.transactions.transaction  
  
object Artists : IntIdTable() {  
 val name = varchar("name", 20)  
}  
  
class Artist(id: EntityID<Int>) : IntEntity(id) {  
 companion object : IntEntityClass<Artist>(Artists) {  
 fun findOrCreate(name: String): Artist {  
 return transaction **{** Artist.find **{** Artists.name eq name **}**.firstOrNull() ?: Artist.new **{** this.name = name  
 **}  
 }** }  
 }  
  
 var name by Artists.name  
 val tracks by MusicTrack optionalReferrersOn MusicTracks.artist  
  
 override fun toString(): String {  
 return name  
 }  
}

**Код программы**

Объект DatabaseManager используется для установления соединения с БД и инициализации таблиц:

package dmitry.tpo2.client  
  
import dmitry.tpo2.entity.Artists  
import dmitry.tpo2.entity.MusicTracks  
import dmitry.tpo2.entity.Users  
import org.jetbrains.exposed.sql.Database  
import org.jetbrains.exposed.sql.SchemaUtils.create  
import org.jetbrains.exposed.sql.transactions.transaction  
  
object DatabaseManager {  
 const val URL = "jdbc:postgresql://balarama.db.elephantsql.com:5432/yavpbler"  
 const val DRIVER = "org.postgresql.Driver"  
 const val USER = "yavpbler"  
 const val PASSWORD = "mk9TrGVmf9BvwhxhGFOYr1VDJ1v3ot7S"  
  
 lateinit var database: Database  
 private set  
  
 fun initDatabase() {  
 database = Database.connect(URL, DRIVER, USER, PASSWORD)  
 transaction **{** create(Users)  
 create(Artists)  
 create(MusicTracks)  
 **}** }  
}

Точкой входа в приложение служит класс AppClient:

package dmitry.tpo2  
  
import dmitry.tpo2.client.AppClient  
  
fun main(args: Array<String>) {  
 val client = AppClient()  
 client.start()  
}

package dmitry.tpo2.client  
  
import dmitry.tpo2.entity.User  
import dmitry.tpo2.pages.AbstractPage  
import dmitry.tpo2.pages.LoginPage  
import dmitry.tpo2.pages.SignUpPage  
import org.jetbrains.exposed.sql.transactions.transaction  
  
class AppClient {  
 var page: AbstractPage? = null  
 private set  
  
 val hasUsers : Boolean  
 get() {  
 return transaction **{** User.count() > 0  
 **}** }  
  
 fun start() {  
 try {  
 DatabaseManager.initDatabase()  
 page = if (hasUsers) LoginPage else SignUpPage  
 interact()  
 } catch (e: Exception) {  
 println("Ошибка при инициализации клиента: ")  
 e.printStackTrace()  
 }  
 }  
  
 fun interact() {  
 while (page != null) {  
 page = page!!.interact()  
 }  
 }  
  
}

Приложение построено на страницах, которые расширяют абстрактный класс AbstractPage. Метод interact() взаимодействует с пользователем и определяет, на какую страницу необходимо перейти дальше. Приложение завершает работу, когда будет возвращён null (см. класс AppClient выше)

package dmitry.tpo2.pages  
  
abstract class AbstractPage {  
 protected abstract val pageTitle: String  
  
 abstract fun runInteractionLogic(): AbstractPage?  
  
 fun interact() : AbstractPage? {  
 println("\n--------------$pageTitle--------------")  
 return runInteractionLogic()  
 }  
  
 fun readUserInput(default: String = ""): String {  
 return readLine() ?: default  
 }  
  
 fun readInt(): Int? {  
 val input = readLine()  
 return (input ?: "").toIntOrNull()  
 }  
  
 fun anykey() {  
 println("\nНажмите Enter, чтобы продолжить")  
 readLine()  
 }  
}

При запуске необходимо войти либо зарегистрироваться, если в системе нет аккаунтов:

package dmitry.tpo2.pages  
  
import dmitry.tpo2.entity.User  
import dmitry.tpo2.entity.Users  
import org.jetbrains.exposed.dao.EntityID  
import org.jetbrains.exposed.sql.transactions.transaction  
  
object LoginPage : AbstractPage() {  
 override val pageTitle = "Вход"  
  
 fun getUser(): User {  
 println("Введите логин: ")  
 var user: User?  
 do {  
 user = transaction **{** val username = readUserInput()  
 User.find **{** Users.username eq username **}**.firstOrNull()  
 **}** if (user == null) {  
 println("Пользователь с таким логином не найден. Попробуйте ещё раз: ")  
 }  
 } while(user == null)  
  
 return user  
 }  
  
 fun tryEnterPassword(user: User): Boolean {  
 val password = readUserInput()  
 return user.hashedPassword == password.hashCode()  
 }  
  
 fun enterPassword(user: User) {  
 println("Введите пароль: ")  
 do {  
 val success = tryEnterPassword(user)  
 if (!success) {  
 println("Неверный пароль. Попробуйте ещё раз: ")  
 }  
 } while(!success)  
 }  
  
 override fun runInteractionLogic(): AbstractPage {  
 val user = getUser()  
 enterPassword(user)  
  
 return MenuPage  
 }  
}

package dmitry.tpo2.pages  
  
import dmitry.tpo2.entity.User  
import dmitry.tpo2.entity.Users  
import org.jetbrains.exposed.sql.transactions.transaction  
  
object SignUpPage : AbstractPage() {  
 override val pageTitle = "Регистрация"  
  
 fun inputUsername(): String {  
 var username: String  
 do {  
 username = readUserInput().trim()  
 if (username.isBlank()) {  
 println("Имя пользователя не должно быть пустым. Попробуйте ещё раз:")  
 }  
 } while (username.isBlank())  
 return username  
 }  
  
 fun isUsernameTaken(username: String): Boolean {  
 val user = transaction **{** User.find **{** Users.username eq username **}**.firstOrNull()  
 **}** return user != null  
 }  
  
 fun validatePassword(password: String): Boolean {  
 val longEnough = password.length >= 6  
 val hasLetters = password.any **{** it.isLetter() **}** val hasDigits = password.any **{** it.isDigit() **}** return longEnough && hasLetters && hasDigits  
 }  
  
 fun createUser(username: String, password: String) {  
 transaction **{** User.new **{** this.username = username  
 this.hashedPassword = password.hashCode()  
 **}  
 }** }  
  
 override fun runInteractionLogic(): AbstractPage {  
 lateinit var username: String  
 lateinit var password: String  
  
 println("Введите имя пользователя:")  
 do {  
 username = inputUsername()  
 val taken = isUsernameTaken(username)  
 if (taken) {  
 println("Пользователь с таким логином уже существует. Попробуйте ещё раз: ")  
 }  
 } while(taken)  
  
 println("Введите пароль:")  
 do {  
 password = readUserInput()  
 val invalid = validatePassword(password)  
 if (invalid) {  
 println("Пароль должен содержать минимум 6 символов, включая хотя бы одну букву и одну цифру. \nПопробуйте ещё раз: ")  
 }  
 } while(invalid)  
  
 createUser(username, password)  
  
 return MenuPage  
 }  
}

После входа открывается консольное меню, позволяющее выбрать один из пунктов:

package dmitry.tpo2.pages  
  
object MenuPage : AbstractPage() {  
 private const val menuItems =  
 "[1] Все композиции\n" +  
 "[2] Поиск\n" +  
 "[3] Добавить композицию\n" +  
 "[4] Удалить композицию\n" +  
 "[5] Информация о композиции\n" +  
 "[6] О программе\n" +  
 "[Другое число] Выход"  
  
 override val pageTitle = "Меню"  
  
 override fun runInteractionLogic(): AbstractPage? {  
 println("Введите число для выбора пункта меню: ")  
 println(menuItems)  
 val item = readInt()  
 return when (item) {  
 1 -> TrackListPage  
 2 -> SearchPage  
 3 -> NewTrackPage  
 4 -> DeletePage  
 5 -> MusicTrackInfoPage  
 6 -> AboutPage  
 else -> null  
 }  
 }  
}

Пункт №1 – вывод всех композиций:

package dmitry.tpo2.pages  
  
import dmitry.tpo2.entity.MusicTrack  
import org.jetbrains.exposed.sql.SizedIterable  
import org.jetbrains.exposed.sql.transactions.transaction  
  
object TrackListPage : AbstractPage() {  
 override val pageTitle = "Каталог"  
  
 fun getTracks(): SizedIterable<MusicTrack> {  
 return transaction **{** MusicTrack.all()  
 **}** }  
  
 fun printList(list: SizedIterable<MusicTrack>) {  
 transaction **{** if (list.count() > 0) {  
 list.forEach(::println)  
 } else {  
 println("Ничего не найдено!")  
 }  
 **}** }  
  
 override fun runInteractionLogic(): AbstractPage {  
 val list = getTracks()  
 printList(list)  
  
 anykey()  
 return MenuPage  
 }  
}

Пункт №2 – поиск. Доступны три вида поиска – по названию, по исполнителю и по году:

package dmitry.tpo2.pages  
  
import dmitry.tpo2.entity.Artists  
import dmitry.tpo2.entity.MusicTrack  
import dmitry.tpo2.entity.MusicTracks  
import org.jetbrains.exposed.sql.SizedIterable  
import org.jetbrains.exposed.sql.SqlExpressionBuilder.eq  
import org.jetbrains.exposed.sql.SqlExpressionBuilder.isNotNull  
import org.jetbrains.exposed.sql.SqlExpressionBuilder.like  
import org.jetbrains.exposed.sql.and  
import org.jetbrains.exposed.sql.lowerCase  
import org.jetbrains.exposed.sql.select  
import org.jetbrains.exposed.sql.transactions.transaction  
  
object SearchPage : AbstractPage() {  
 override val pageTitle = "Поиск"  
  
 fun searchByName(name: String): SizedIterable<MusicTrack> {  
 val lowercaseName = name.toLowerCase()  
 return transaction **{** MusicTrack.find **{** MusicTracks.name.lowerCase() like "%$lowercaseName%" **}  
 }** }  
  
 private fun searchByArtist(artist: String): SizedIterable<MusicTrack> {  
 val lowercaseArtist = artist.toLowerCase()  
 return transaction **{** val query = (MusicTracks innerJoin Artists).slice(MusicTracks.columns).select **{** MusicTracks.artist.isNotNull() and (Artists.name.lowerCase() like "%$lowercaseArtist%")  
 **}** MusicTrack.wrapRows(query)  
 **}** }  
  
 private fun searchByYear(year: Int): SizedIterable<MusicTrack> {  
 return transaction **{** MusicTrack.find **{** MusicTracks.year.isNotNull() and (MusicTracks.year eq year) **}  
 }** }  
  
 private fun printList(list: SizedIterable<MusicTrack>) {  
 transaction **{** if (list.count() > 0) {  
 list.forEach(::println)  
 } else {  
 println("По данному запросу ничего не найдено!")  
 }  
 **}** }  
  
 override fun runInteractionLogic(): AbstractPage {  
 println("Выберите способ поиска: ")  
 println("[1] По названию")  
 println("[2] По исполнителю")  
 println("[3] По году")  
  
 val item = readInt()  
 val list: SizedIterable<MusicTrack>? = when (item) {  
 1 -> {  
 println("Введите название: ")  
 val name = readUserInput().trim()  
 searchByName(name)  
 }  
 2 -> {  
 println("Введите исполнителя: ")  
 val artist = readUserInput().trim()  
 searchByArtist(artist)  
 }  
 3 -> {  
 println("Введите год: ")  
 val year = readInt()  
 if (year == null || year < 0) {  
 println("Ошибка: год должен быть корректным положительным числом.")  
 null  
 } else {  
 searchByYear(year)  
 }  
 }  
 else -> {  
 println("Ошибка: некорректный пункт меню.")  
 null  
 }  
 }  
  
 if (list != null) {  
 printList(list)  
 }  
  
 anykey()  
 return MenuPage  
 }  
}

Пункт №3 – добавление новой композиции:

package dmitry.tpo2.pages  
  
import dmitry.tpo2.entity.Artist  
import dmitry.tpo2.entity.MusicTrack  
import org.jetbrains.exposed.sql.transactions.transaction  
  
object NewTrackPage : AbstractPage() {  
 override val pageTitle = "Добавить аудиозапись"  
  
 fun addTrack(name: String, artist: String, year: Int?) {  
 transaction **{** MusicTrack.new **{** this.name = name  
 this.artist = Artist.findOrCreate(artist)  
 this.year = year  
 **}  
 }** }  
  
 override fun runInteractionLogic(): AbstractPage {  
 return try {  
 println("Введите название: ")  
 val name = readUserInput("Без названия")  
 println("Введите исполнителя: ")  
 val artist = readUserInput("Неизвестный исполнитель")  
 println("Введите год: ")  
 val year = readInt()  
 if (year != null && year < 0) throw IllegalArgumentException("год не может быть отрицательным!")  
  
 addTrack(name, artist, year)  
  
 println("Композиция успешно добавлена")  
 anykey()  
 MenuPage  
 } catch (e: Exception) {  
 println("Ошибка при вводе: ${e.localizedMessage}")  
 anykey()  
 this  
 }  
  
 }  
}

Пункт №4 – удаление композиции:

package dmitry.tpo2.pages  
  
import dmitry.tpo2.entity.MusicTrack  
import org.jetbrains.exposed.sql.transactions.transaction  
  
object DeletePage : AbstractPage() {  
 override val pageTitle = "Удаление"  
  
 fun tryDeleteTrack(id: Int): Boolean {  
 return transaction **{** val track = MusicTrack.findById(id)  
 if (track != null) {  
 track.delete()  
 true  
 } else false  
 **}** }  
  
 override fun runInteractionLogic(): AbstractPage {  
 println("Введите ID записи для удаления: ")  
  
 val id = readInt()  
 val deleted = tryDeleteTrack(id ?: 0)  
 if (deleted) {  
 println("Запись удалена.")  
 } else {  
 println("Запись с таким ID не найдена.")  
 }  
 anykey()  
 return MenuPage  
 }  
}

Пункт №5 – информация о композиции:

package dmitry.tpo2.pages  
  
import dmitry.tpo2.entity.MusicTrack  
import org.jetbrains.exposed.sql.transactions.transaction  
  
object MusicTrackInfoPage : AbstractPage() {  
 override val pageTitle = "Информация об аудиозаписи"  
  
 fun getTrack(id: Int?): MusicTrack? {  
 if (id == null) return null  
 return transaction **{** MusicTrack.findById(id)  
 **}** }  
  
 fun printTrack(track: MusicTrack?) {  
 if (track != null) {  
 transaction **{** println(track)  
 **}** } else println("Ошибка: аудиозапись с таким ID не найдена.")  
 }  
  
 override fun runInteractionLogic(): AbstractPage {  
 println("Введите ID: ")  
 val id = readInt()  
 val track = getTrack(id)  
 printTrack(track)  
  
 anykey()  
 return MenuPage  
 }  
}

Пункт № 6 – о программе

package dmitry.tpo2.pages  
  
object AboutPage : AbstractPage() {  
 override val pageTitle = "О программе"  
  
 override fun runInteractionLogic(): AbstractPage {  
 println(" - Данное ПО доступно по лицензии WTFPL\n" +  
 " - Делайте что хотите\n" +  
 " - Никакие права не защищены\n")  
 anykey()  
 return MenuPage  
 }  
}

**Тесты**

Тесты клиента: проверяем, что при наличии пользователя предлагается вход, а при отсутствии – регистрация:

package dmitry.tpo2.client  
  
import com.nhaarman.mockitokotlin2.doNothing  
import com.nhaarman.mockitokotlin2.doReturn  
import com.nhaarman.mockitokotlin2.spy  
import com.nhaarman.mockitokotlin2.whenever  
import dmitry.tpo2.pages.LoginPage  
import dmitry.tpo2.pages.SignUpPage  
import io.kotlintest.specs.StringSpec  
import org.junit.Assert.assertTrue  
  
class AppClientTests : StringSpec(**{** val spy = spy(AppClient())  
 doNothing().whenever(spy).interact()  
  
 "Returns SignUpPage when there are no users" **{** doReturn(false).whenever(spy).hasUsers  
 spy.start()  
 assertTrue("Page was ${spy.page!!::class.simpleName}", spy.page is SignUpPage)  
 **}** "Returns LoginPage when there is at least one user" **{** doReturn(true).whenever(spy).hasUsers  
 spy.start()  
 assertTrue("Page was ${spy.page!!::class.simpleName}", spy.page is LoginPage)  
 **}  
}**)

Тесты менеджера БД: проверяем, что инициализация соединения проходит успешно:

package dmitry.tpo2.client  
  
import io.kotlintest.shouldNotThrowAny  
import io.kotlintest.specs.StringSpec  
  
class DatabaseManagerTests : StringSpec(**{** "Connects and creates tables" **{** shouldNotThrowAny **{** DatabaseManager.initDatabase()  
 **}  
 }  
}**)

Тесты страницы входа: проверяем функциональность входа, вводим неправильные пароли

package dmitry.tpo2.pages  
  
import com.nhaarman.mockitokotlin2.\*  
import dmitry.tpo2.entity.User  
import io.kotlintest.specs.StringSpec  
import org.junit.Assert.assertTrue  
  
class LoginPageTests : StringSpec(**{** val spy = spy(LoginPage)  
  
 val password = "qwerty123456"  
 val fakeUser = mock<User> **{** on **{** ::hashedPassword.get() **}** doReturn password.hashCode()  
 **}** "Returns MenuPage and validates password" **{** doReturn(fakeUser).whenever(spy).getUser()  
 whenever(spy.readUserInput())  
 .thenReturn("")  
 .thenReturn("123456")  
 .thenReturn("йцукен")  
 .thenReturn("QwErTy1@3$5^")  
 .thenReturn(password)  
  
 val page = spy.runInteractionLogic()  
 assertTrue("Page was ${page::class.simpleName}", page is MenuPage)  
  
 verify(spy, times(5)).readUserInput()  
 **}  
}**)

Тесты страницы регистрации: проверяем валидацию пароля

package dmitry.tpo2.pages  
  
import com.nhaarman.mockitokotlin2.doNothing  
import com.nhaarman.mockitokotlin2.doReturn  
import com.nhaarman.mockitokotlin2.spy  
import com.nhaarman.mockitokotlin2.whenever  
import io.kotlintest.specs.StringSpec  
import org.junit.Assert.assertFalse  
import org.junit.Assert.assertTrue  
import org.mockito.ArgumentMatchers.anyString  
  
class SignUpPageTests : StringSpec(**{** val spy = spy(SignUpPage)  
 doNothing().whenever(spy).createUser(anyString(), anyString())  
  
 "Password is too short" **{** val success = spy.validatePassword("q123")  
 assertFalse(success)  
 **}** "Password has no letters" **{** val success = spy.validatePassword("1234567890")  
 assertFalse(success)  
 **}** "Password has no digits" **{** val success = spy.validatePassword("qwertyuiop")  
 assertFalse(success)  
 **}** "Valid password" **{** val success = spy.validatePassword("qwerty123456")  
 assertTrue(success)  
 **}  
}**)

Тесты меню: проверяем переходы

package dmitry.tpo2.pages  
  
import com.nhaarman.mockitokotlin2.spy  
import com.nhaarman.mockitokotlin2.whenever  
import io.kotlintest.specs.StringSpec  
import org.junit.Assert.assertNull  
import org.junit.Assert.assertTrue  
  
class MenuPageTests : StringSpec(**{** val spy = spy(MenuPage)  
  
 fun testMenuItem(id: Int, requiredPage: AbstractPage) {  
 whenever(spy.readInt()).thenReturn(id)  
 val page = spy.runInteractionLogic()  
 assertTrue("Page was ${page!!::class.simpleName}", page::class == requiredPage::class)  
 }  
  
 "Track list page" **{** testMenuItem(1, TrackListPage)  
 **}** "Search page" **{** testMenuItem(2, SearchPage)  
 **}** "New track page" **{** testMenuItem(3, NewTrackPage)  
 **}** "Delete page" **{** testMenuItem(4, DeletePage)  
 **}** "Music track info" **{** testMenuItem(5, MusicTrackInfoPage)  
 **}** "About page" **{** testMenuItem(6, AboutPage)  
 **}** "Exit" **{** whenever(spy.readInt()).thenReturn(100)  
 val page = spy.runInteractionLogic()  
 assertNull(page)  
 **}  
}**)

Тесты списка всех композиций: проверяем переход

package dmitry.tpo2.pages  
  
import com.nhaarman.mockitokotlin2.\*  
import dmitry.tpo2.client.DatabaseManager  
import dmitry.tpo2.entity.MusicTrack  
import io.kotlintest.specs.StringSpec  
import org.jetbrains.exposed.sql.SizedIterable  
import org.jetbrains.exposed.sql.emptySized  
import org.junit.Assert.assertTrue  
import org.mockito.ArgumentMatchers  
import org.mockito.Matchers  
  
  
  
class TrackListPageTests : StringSpec(**{** DatabaseManager.initDatabase()  
  
 val spy = spy(TrackListPage) **{** on **{** getTracks() **}** doReturn emptySized()  
 **}** doNothing().whenever(spy).anykey()  
  
 "Returns MenuPage" **{** val page = spy.runInteractionLogic()  
 assertTrue("Page was ${page::class.simpleName}", page is MenuPage)  
 **}  
}**)

Тесты страницы поиска: проверяем переход

package dmitry.tpo2.pages  
  
import com.nhaarman.mockitokotlin2.\*  
import dmitry.tpo2.client.DatabaseManager  
import io.kotlintest.specs.StringSpec  
import org.jetbrains.exposed.sql.emptySized  
import org.junit.Assert.assertTrue  
import org.mockito.ArgumentMatchers.anyString  
  
class SearchPageTests : StringSpec(**{** DatabaseManager.initDatabase()  
  
 val spy = spy(SearchPage) **{** on **{** searchByName(anyString()) **}** doReturn emptySized()  
 **}** doNothing().whenever(spy).anykey()  
  
 "Returns MenuPage" **{** whenever(spy.readInt()).thenReturn(4)  
  
 val page = spy.runInteractionLogic()  
 assertTrue("Page was ${page::class.simpleName}", page is MenuPage)  
 **}  
}**)

Тесты страницы добавления: проверяем переход

package dmitry.tpo2.pages  
  
import com.nhaarman.mockitokotlin2.doNothing  
import com.nhaarman.mockitokotlin2.doReturn  
import com.nhaarman.mockitokotlin2.spy  
import com.nhaarman.mockitokotlin2.whenever  
import io.kotlintest.specs.StringSpec  
import org.junit.Assert  
import org.mockito.ArgumentMatchers.anyString  
import org.mockito.ArgumentMatchers.anyInt  
  
class NewTrackPageTests : StringSpec(**{** val spy = spy(NewTrackPage)  
 doNothing().whenever(spy).addTrack(anyString(), anyString(), anyInt())  
 doNothing().whenever(spy).anykey()  
  
 whenever(spy.readUserInput())  
 .thenReturn("Test track")  
 .thenReturn("Test artist")  
  
 "Returns MenuPage on success" **{** whenever(spy.readInt()).doReturn(2007)  
 val page = spy.runInteractionLogic()  
 Assert.assertTrue("Page was ${page::class.simpleName}", page is MenuPage)  
 **}** "Returns itself on failure" **{** whenever(spy.readInt()).doReturn(-2007)  
 val page = spy.runInteractionLogic()  
 Assert.assertTrue("Page was ${page::class.simpleName}", page is NewTrackPage)  
 **}  
}**)

Тесты страницы удаления: проверяем переход

package dmitry.tpo2.pages  
  
import com.nhaarman.mockitokotlin2.doNothing  
import com.nhaarman.mockitokotlin2.doReturn  
import com.nhaarman.mockitokotlin2.spy  
import com.nhaarman.mockitokotlin2.whenever  
import io.kotlintest.specs.StringSpec  
import org.junit.Assert.assertTrue  
import org.mockito.ArgumentMatchers.anyInt  
  
class DeletePageTests : StringSpec(**{** val spy = spy<DeletePage> **{** on **{** readInt() **}** doReturn 20  
 **}** doReturn(true).whenever(spy).tryDeleteTrack(anyInt())  
 doNothing().whenever(spy).anykey()  
  
 "Returns to menu" **{** val page = spy.runInteractionLogic()  
 assertTrue("Page was ${page::class.simpleName}", page is MenuPage)  
 **}  
}**)

Тесты страницы подробной информации: проверяем переход

package dmitry.tpo2.pages  
  
import com.nhaarman.mockitokotlin2.\*  
import dmitry.tpo2.entity.MusicTrack  
import io.kotlintest.specs.StringSpec  
import org.jetbrains.exposed.sql.SizedIterable  
import org.junit.Assert.assertTrue  
import org.mockito.ArgumentMatchers  
import org.mockito.ArgumentMatchers.anyInt  
  
class MusicTrackInfoPageTests : StringSpec(**{** val spy = spy(MusicTrackInfoPage) **{** on **{** readInt() **}** doReturn 10  
 **}** doReturn(null).whenever(spy).getTrack(anyInt())  
 doNothing().whenever(spy).anykey()  
  
 "Returns MenuPage" **{** val page = spy.runInteractionLogic()  
 assertTrue("Page was ${page::class.simpleName}", page is MenuPage)  
 **}  
}**)

Тесты страницы «о программе»: проверяем переход

package dmitry.tpo2.pages  
  
import com.nhaarman.mockitokotlin2.doNothing  
import com.nhaarman.mockitokotlin2.spy  
import com.nhaarman.mockitokotlin2.whenever  
import io.kotlintest.specs.StringSpec  
import org.junit.Assert.assertTrue  
  
class AboutPageTests : StringSpec(**{** val spy = spy<AboutPage>()  
 doNothing().whenever(spy).anykey()  
  
 "Returns to main menu" **{** val page = spy.runInteractionLogic()  
 assertTrue("Page was ${page::class.simpleName}", page is MenuPage)  
 **}  
}**)

**Выводы**

Разработка через тестирование позволяет разрабатывать приложения, которые, во-первых, подходят для тестирования в силу своей архитектуры, а во-вторых, покрыты тестами почти на 100%. TDD позволяет сосредоточиться на требованиях заказчика и отсечь всё лишнее по принципу KISS.

К сожалению, TDD обладает и рядом недостатков, самый очевидный из которых – затраты времени на написание тестов и их дальнейшее поддержание. Кроме того, в некоторых случаях автоматическое тестирование крайне сложно или вовсе невозможно осуществить: например, архитектура фреймворка Exposed совершенно не подходит для мокинга. Наконец, код программы и тестов пишет один и тот же человек, так что если он неправильно поймёт ТЗ, то может допустить ошибку и там, и там.