Laboratorium 10
Testy jednostkowe z użyciem narzędzia JUnit

Cel laboratorium:
Nabycie umiejętności tworzenia testów jednostkowych za pomocą narzędzia JUnit.

1. Należy wykonać test jednostkowy metod klasy, która stanowi klasę końcową w łańcuchu powiązań na diagramie klas lub/i może być powiązana w relacji 1 do 0..1 z inną klasą – podobnie jak klasa typu ***Factory*** lub klasy ***Book, Client***. Należy zastosować w metodach testowych metody klasy ***Assert*** z pakietu org.junit biblioteki ***JUnit 4.12*** oraz adnotacje: ***Test, Parameter, Parameters, RunWith(Parameterized.class), Rule***.
   Dane wzorcowe, wykorzystywane do weryfikacji wyników testowanych metod za pomocą metod klasy ***Assert*** należy umieścić w dodatkowej klasie, podobnie jak klasa ***Data*** z p.2.0 **Dodatku 1**. Przykłady testów podano w **Dodatku 1**.
   W tabelce poniżej podano in**for**mację dotyczącą wyboru metod do testowania oraz przykładów rozwiązań.

Przykłady testowanych klas			
Factory	Client	Book	Reservation

2. Należy wykonać test jednostkowy metod klasy, która stanowi klasę w łańcuchu powiązań na diagramie klas lub/i może być powiązana w relacji „1 do 1" lub „1 do 1..*" z inną/innymi klasami – podobnie jak klasy typu ***TiTleBook, TitleBookRead, Facade.*** Należy zastosować w metodach testowych metody klasy ***Assert*** z biblioteki ***JUnit 4.12*** oraz adnotacje: ***Test, Parameter, Parameters, RunWith(Parameterized.class), FixMethodOrder(MethodSorters.NAME_ASCENDING), Rule***.
   Przykłady testów podano w **Dodatku 1**.
   Dane wzorcowe wykorzystywane do weryfikacji wyników testowanych metod za pomocą metod klasy ***Assert***, należy umieścić w dodatkowej klasie (zdefiniowanej w p. 2), podobnie jak klasa ***Data*** z p.2.1 **Dodatku 1**. Kryterium wyboru metod powinno uwzględniać fakt, że metody wybrane w p.1 są wywoływane w metodach klas wybranych w p.2.
   Poniżej, w tabelce poniżej podano in**for**mację dotyczącą wyboru metod do testowania oraz przykładów rozwiązań.

Przykłady testowanych klas		
Facade	TitleBook	TitleBookRead

3. Należy wykonać zestawy testów, podobnie jak pokazano w p.2.7 **Dodatku 1** stosując adnotację ***Category*** w klasach z metodami testującymi, wykonanych w p. 1, 2 oraz ***@Categories.SuiteClasses, @RunWith(Categories.class), @Categories.IncludeCategory, Categories.ExcludeCategory***.

Dodatek 1

Testy jednostkowe oprogramowania "Wypożyczalnia książek" - dodatek 1 zawiera ważne informacje wspomagające tworzenie testów.

1. **Dodanie generowania wyjątku** w przypadku niepoprawnej wartości pierwszego elementu tablicy ***data*** jako parametru metody ***createTitlebook*** klasy ***Factory***. **Pełna walidacja poprawności danych wejściowych powinna być realizowana przez warstwę klienta aplikacji!**

```
package subbusinesstier;

import java.time.LocalDate;
import java.util.IllegalFormatCodePointException;
import subbusinesstier.entities.Book;
import subbusinesstier.entities.Client;
import subbusinesstier.entities.Reservation;
import subbusinesstier.entities.TitleBook;
import subbusinesstier.entities.TitleBookRead;

public class Factory {

  public TitleBook createTitleBook(String data[]) {
    TitleBook titleBook = null;
    switch (Integer.parseInt(data[0])) //what_title_book_type
    {
      case 0:
        titleBook = new TitleBook(); //TTitle_book object for searching
        titleBook.setISBN(data[1]);
        break;
      case 1:
        titleBook = new TitleBook(); //TTitle_book object for persisting
        titleBook.setAuthor(data[1]);
        titleBook.setTitle(data[2]);
        titleBook.setISBN(data[3]);
        titleBook.setPublisher(data[4]);
        break;
      case 2:
        TitleBookRead title_book1 = new TitleBookRead();//TTitle_book_on_tape object for searching
        title_book1.setISBN(data[1]);
        title_book1.setActor(data[2]);
        titleBook = title_book1;
        break;
      case 3:
        TitleBookRead title_book2 = new TitleBookRead();//TTitle_book_on_tape object for persisting
        title_book2.setAuthor(data[1]);
        title_book2.setTitle(data[2]);
        title_book2.setISBN(data[3]);
        title_book2.setPublisher(data[4]);
        title_book2.setActor(data[5]);
        titleBook = title_book2;
        break;
      default:
        throw new IllegalFormatCodePointException(0); //generowanie wyjątku z powodu niepoprawnej
                                                      //wartości elementu tablicy dane o indeksie 0.
    }
    return titleBook;
  }
```

```
public Book createBook(String data[]) {
      Book book = null;
      switch (Integer.parseInt(data[0])) //what_book_type
      {
         case 0:
            book = new Book();//TBook object for persisting
            book.setNumber(Integer.parseInt(data[1]));
            break; }
      return book;
   }
   public Client createClient(String data[]) {
      Client client = null;
      switch (Integer.parseInt(data[0])) //what_book_type
      {
         case 0:
            client = new Client();
            client.setNumberCard(Integer.parseInt(data[1]));
            break;
         case 1:
            client = new Client();
            client.setName(data[1]);
            client.setNumberCard(Integer.parseInt(data[2]));
            break; }
      return client;
   }
   public Reservation createReservation(int number, LocalDate date) {
      Reservation reservation = new Reservation();
      reservation.setNumber(number);
      reservation.setDate(date);
      return reservation;
   }
}
```

2. Dodanie konstruktorów z parametrami do testowanych klas:

```
public Book(int number) {
      reservations = new ArrayList();
      this.number = number;
   }
public Client(int numberCard, String name) {
      this.numberCard = numberCard;
      this.name = name;
      reservations=new ArrayList();
   }
 public Reservation(int number, LocalDate date) {
      this.number = number;
      this.date = date;
   }
public TitleBook(String publisher, String ISBN, String title, String author) {
       books = new ArrayList();
      this.publisher = publisher;
      this.ISBN = ISBN;
      this.title = title;
      this.author = author;
   }
public TitleBookRead(String actor, String publisher, String ISBN, String title, String author) {
      super(publisher, ISBN, title, author);
      this.actor = actor;
   }
```

2. Testy jednostkowe z wykorzystaniem narzędzia *JUnit 4.12*

2.0. Definicja danych wzorcowych -

```
package testdata;

import java.time.LocalDate;
import java.time.Month;
import java.util.Arrays;
import java.util.List;
import subbusinesstier.entities.Book;
import subbusinesstier.entities.Client;
import subbusinesstier.entities.Reservation;
import subbusinesstier.entities.TitleBook;
import subbusinesstier.entities.TitleBookRead;

public class Data {

   public String titledata[][] = {
      {"1", "Author1", "Title1", "ISBN1", "Publisher1"},
      {"1", "Author2", "Title2", "ISBN2", "Publisher2"},
      {"1", "Author3", "Title3", "ISBN3", "Publisher3"},
      {"3", "Author1", "Title1", "ISBN1", "Publisher1", "Actor1"},
      {"3", "Author2", "Title2", "ISBN2", "Publisher2", "Actor2"},
      {"3", "Author4", "Title4", "ISBN4", "Publisher4", "Actor4"},

      {"0", "ISBN1"}, {"0", "ISBN2"}, {"0", "ISBN3"},
      {"2", "ISBN1", "Actor1"},{"2", "ISBN2", "Actor2"},{"2", "ISBN4", "Actor4"},

      {"4", "Author4", "Title4", "ISBN4", "Publisher4", "Actor4"}
   };

   public TitleBook titles[] = {
      new TitleBook("Publisher1", "ISBN1", "Title1", "Author1"),
      new TitleBook("Publisher2", "ISBN2", "Title2", "Author2"),
      new TitleBook("Publisher3", "ISBN3", "Title3", "Author3"),
      new TitleBookRead("Actor1", "Publisher1", "ISBN1", "Title1", "Author1"),
      new TitleBookRead("Actor2", "Publisher2", "ISBN2", "Title2", "Author2"),
      new TitleBookRead("Actor4", "Publisher4", "ISBN4", "Title4", "Author4")};


   public String bookdata[][] = {{"0", "1"}, {"0", "2"}};
   public Book books[] = {new Book(1), new Book(2)};

   public String[][] clientdata = {{"1", "Klient1", "1"}, {"1", "Klient2", "2"}, {"0", "1"}, {"0", "2"}, {"0", "3"}};
   public Client[] clients = {new Client(1, "KLient1"), new Client(2, "Klient2"), new Client(1, "KLient1"),
                     new Client(2, "KLient2"), new Client(3, "KLient3"),};

   public static int number = 0;

   public LocalDate[] dates = {LocalDate.of(2018, Month.JANUARY, 20), LocalDate.of(2018, Month.MARCH, 20),
                     LocalDate.of(2018, Month.MAY, 20)};
   public Reservation reservations[] = {new Reservation(0, dates[0]), new Reservation(1, dates[0]),
                           new Reservation(2, dates[0]), new Reservation(3, dates[1])};

   public List<Reservation> reservationList = Arrays.asList(reservations);
```

```
 public String databooks1[]={"\nTitle: Title1 Author: Author1 ISBN: ISBN1 Publisher: Publisher1 Number: 1",
                    "\nTitle: Title1 Author: Author1 ISBN: ISBN1 Publisher: Publisher1 Number: 2"};
   public String databooks2[]={"\nTitle: Title4 Author: Author4 ISBN: ISBN4 Publisher: Publisher4 Actor: Actor4
                Number: 1",
                    "\nTitle: Title4 Author: Author4 ISBN: ISBN4 Publisher: Publisher4 Actor: Actor4 Number: 2"};
   public String databooks3[]={"\nTitle: Title1 Author: Author1 ISBN: ISBN2 Publisher: Publisher1 Number: 1",
                //ISBN2
                    "\nTitle: Title1 Author: Author1 ISBN: ISBN1 Publisher: Publisher1 Number: 2"};
}
```

2.1. Test jednostkowy klasy ***Factory*** **(wynik działania: p.2.7.1, 2.7.3, 2.7.4)** – przykłady prostych testów tworzenia obiektów typu: TitleBook, TitleBookRead, Book, Client, Reservation).
Należy wykonać interfejs pusty o nazwie Test_Control w celu zastosowania adnotacji @Category. Jest wstęp do utworzenia zestawu testów.

Zastosowanie adnotacji
@Test
@BeforeClass
@Category
@Rule

Zastosowanie metod
static public void assertEquals(Object expected, Object actual) //k1.1
ExpectedException //k1.2

```
package subbusinesstier;

import categories.Test_Control;
import java.util.IllegalFormatCodePointException;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.Rule;
import org.junit.experimental.categories.Category;
import org.junit.rules.ExpectedException;
import subbusinesstier.entities.Book;
import subbusinesstier.entities.Client;
import subbusinesstier.entities.Reservation;
import subbusinesstier.entities.TitleBook;
import testdata.Data;

@Category({Test_Control.class})
public class FactoryTest {

 static Data data;
  @Rule
  public ExpectedException exception = ExpectedException.none(); //definicja obiektu odpowiedzialnego
          //za zachowanie metody testującej podczas generowania wyjątku przez testowaną metodę

 @BeforeClass
  public static void setUpClass() {
   data=new Data();
  }

  @Test
  public void testCreateTitleBook() {
    System.out.println("createTitleBook");
     Factory instance = new Factory();
     for (int i = 0; i < 6; i++) {
        TitleBook result = instance.createTitleBook(data.titledata[i]);
        assertEquals(data.titles[i], result);        //k1.1 – test poprawności tworzonych tytułów
     }
     exception.expect(IllegalFormatCodePointException.class);    //k1.2 – definicja zachowania metody
     exception.expectMessage("Code point = 0x0");          //testowej podczas testowania generowania
     instance.createTitleBook(data.titledata[12]);        // wyjątku IllegalFormatCodePointException
     }                                                    // przez metodę   createTitleBook
```

```
@Test
 public void testCreateBook() {
    System.out.println("createBook");
    Factory instance = new Factory();
    for (int i = 0; i < 2; i++) {
       Book result = instance.createBook(data.bookdata[i]);
       assertEquals(data.books[i], result);   //k1.1 – test poprawności tworzonych ksiazek
    }  }

@Test
 public void testCreateClient() {
    System.out.println("createClient");
    Factory instance = new Factory();
    for (int i = 0; i < 5; i++) {
       Client result = instance.createClient(data.clientdata[i]);
       assertEquals(data.clients[i], result);  //k1.1 – test poprawności tworzonych klientów
    }
 }

 @Test
 public void testCreateReservation() {
    System.out.println("createReservation");
    Factory instance = new Factory();
    for (int i = 0; i < 3; i++) {
       Reservation result = instance.createReservation(data.number++,data.dates[i]);
       assertEquals(data.reservations[i], result);    //k1.1 – test poprawności tworzonych rezerwacji
    }
 }
}
```

2.2. Test jednostkowy klasy ***Reservation*** **(wynik działania: p.2.7.2, 2.7.4, 2.7.5)** – zastosowanie adnotacji **@Parameter** dla atrybutu ***number1*** i wykonanie metody ***data()*** z adnotacją **@Parameters** powoduje wywołanie 4 razy metody testowej ***testEquals*** z wartościami atrybutu ***number1*** jako indeksu tablicy danych wzorcowych ***reservations***. Wartość indeksu ***number1*** jest wyznaczana z kolejno podstawianych elementów jednowymiarowych tablic zwracanych przez metodę ***data()***. Za pomocą adnotacji **@BeforeClass** wykonano w metodzie ***SetUp*** jednorazowo (przed wykonaniem wszystkich dwóch testów) obiekt typu ***Data*** z danymi wzorcowymi.
Należy wykonać interfejs pusty o nazwie Test_Entity w celu zastosowania adnotacji @Category. Jest wstęp do utworzenia zestawu testów.

Zastosowanie adnotacji
@Test
@BeforeClass
@Category
@Parameter, @Parameters
@RunWith

Zastosowanie metod
static public void assertTrue(boolean condition), //k1.1
static public void assertFalse(boolean condition), //k1.2

```
package subbusinesstier.entities;

import categories.Test_Entity;
import java.util.Arrays;
import java.util.Collection;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.experimental.categories.Category;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import testdata.Data;

@Category({Test_Entity.class})
@RunWith(Parameterized.class)
public class ReservationTest {
   static Data data;

   @BeforeClass
   public static void setUpClass() {
      data=new Data();
  }

   @Parameterized.Parameter
    public int number1;

   @Parameterized.Parameters
   public static Collection<Object[]> data() {
      Object[][] data1 = new Object[][]{{0}, {1}, {2}, {3}};
      return Arrays.asList(data1);
   }
   @Test
   public void testEquals() {
      System.out.println("equals");
     int j=0;
     for (Reservation res:data.reservations) {
        if (number1==j) {
           assertTrue(res.equals(data.reservations[number1])); //k1.1 –test porównania
        } else                                                  // równych rezerwacji
           assertFalse(res.equals(data.reservations[number1])); //k1.2–test porównania
        j++;                                                    //roznych rezerwacji
     }
   }
}
```

2.3. Testy jednostkowe klasy *Book* (wynik działania: p.2.7.2, 2.7.4, 2.7.5) –zastosowanie adnotacji **@Parameter** dla atrybutu ***number1*** i wykonanie metody ***data()*** z adnotacją **@Parameters** powoduje wywołanie 2 razy wszystkich metod testowych z wartościami atrybutu ***number1*** kolejno podstawianych z elementów jednowymiarowych tablic zwracanych przez metodę ***data()***. Za pomocą adnotacji **@BeforeClass** wykonano w metodzie ***SetUp*** jednorazowo (przed wykonaniem wszystkich dwóch testów) obiekt typu ***Data*** z danymi wzorcowymi. W metodzie z adnotacją **@Before** dla kolejnej z 2 grup metod testowych ustawia się wartość obiektu statycznego ***book*** za pomocą elementów tablicy ***books*** obiektów wzorcowych typu ***Book*** **-** wyznaczaną za pomocą atrybutu ***number1****, pełniącego rolę indeksu tablicy*.

Zastosowanie adnotacji
@Test
@BeforeClass
@Before
@Parameter, @Parameters
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
@Category
@RunWith

Zastosowanie metod
static public void assertTrue(boolean condition) // k1.1
static public void assertFalse(boolean condition) // k1.2
static public void assertSame(Object expected, Object actual) //k1.3

```
package subbusinesstier.entities;
import categories.Test_Entity;
import java.util.Arrays;
import java.util.Collection;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.FixMethodOrder;
import org.junit.experimental.categories.Category;
import org.junit.runner.RunWith;
import org.junit.runners.MethodSorters;
import org.junit.runners.Parameterized;
import testdata.Data;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
@Category({Test_Entity.class})
@RunWith(Parameterized.class)
public class BookTest {

   static Data data;
   static Book book;

   @BeforeClass
   public static void setUpClass() {
      data = new Data();
   }
   @Before
   public void setUp() {
      book = data.books[number1];
   }
```

```
@Parameterized.Parameter
public int number1;

@Parameterized.Parameters
public static Collection<Object[]> data() {
   Object[][] data1 = new Object[][]{{0}, {1}};
   return Arrays.asList(data1);
}

@Test
public void testEquals() {
   System.out.println("equals");
   for (int j = number1; j < 2; j++) {
      if (number1 == j) {
         assertTrue(data.books[number1].equals(data.books[j])); //k1.1 –test porównania
      } else // równych produktów
      {
         assertFalse(data.books[number1].equals(data.books[j])); //k1.2–test porównania
      }
   }
}

@Test
public void testAddReservation() {
   System.out.println("addReservation");
   int i = 0;
   for (Reservation res : data.reservations) {
      book.addReservation(res);
      assertSame(res, book.getReservations().get(i));  //k.1.3
      i++;
      assertSame(res.getBook(), book);                 //k.1.3
   }
}

@Test
public void testIsFree() {
   System.out.println("isFree");
   boolean result = true;
   result = book.isFree(data.dates[number1]);
   assertFalse(result);                              //k1.1
   result = book.isFree(data.dates[2]);
   assertTrue(result);                               //k1.2
}
}
```

2.4. Testy jednostkowe klasy Client (wynik działania: p.2.7.2, 2.7.4, 2.7.5)

Zastosowanie adnotacji
@Test
@Category

Zastosowanie metod
static public void assertTrue(boolean condition), //k1.1
static public void assertFalse(boolean condition), //k1.2
static public void assertSame(Object expected, Object actual) //k1.3

```
package subbusinesstier.entities;

import categories.Test_Entity;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.experimental.categories.Category;
import testdata.Data;

@Category({Test_Entity.class})
public class ClientTest {

   static Data data;

  @BeforeClass
   public static void setUpClass() {
       data = new Data();
   }

  @Test
   public void testAddReservation() {
      System.out.println("addReservation");
      Client client=data.clients[0];
      Book book=data.books[0];
         for (int j=0;j<data.dates.length-1;j++) {
            client.addReservation(book, data.dates[j]);
            assertSame(client.getReservations().get(j), book.getReservations().get(j));  //k1.3
            assertSame(client.getReservations().get(j).getClient(), client);              //k1.3
          }
      client.setNumber(0);
     //przywrócenie początkowej wartosci atrybutu number, odpowiedzialnego za numer rezerwacji
   }
   @Test
   public void testEquals() {
      System.out.println("equals");
      for (int j = 0; j < 2; j++) {
         for (int i = j; i < 2; i++) {
            if (i == j) {
               assertTrue(data.clients[i].equals(data.clients[j]));      //k1.1 –test porównania
            } else // równych produktów
               assertFalse(data.clients[i].equals(data.clients[j]));     //k1.2–test porównania
         }
      }
   }
}
```

2.5. Testy jednostkowe klas TitleBook i **TitleBookRead (wynik działania: p.2.7.2, 2.7.4, 2.7.5)** - zastosowanie adnotacji **@Parameter** dla atrybutu ***number1*** i wykonanie metody ***data()*** z adnotacją **@Parameters** powoduje wywołanie 6 razy wszystkich metod testowych z wartościami atrybutu ***number1*** kolejno podstawianych z elementów jednowymiarowych tablic zwracanych przez metodę ***data()***. Za pomocą adnotacji **@BeforeClass** wykonano w metodzie ***SetUp*** jednorazowo (przed wykonaniem wszystkich dwóch testów) obiekt typu ***Data*** z danymi wzorcowymi. W metodzie z adnotacją **@Before** dla kolejnej z 6 grup metod testowych ustawia się wartość obiektu statycznego ***title*** za pomocą elementów tablicy ***titles*** obiektów wzorcowych typu ***TitleBook*****,** wyznaczanych za pomocą indeksu ***number1***. Dzięki narzuceniu kolejności wykonania metod testowych za pomocą adnotacji **@FixMethodOrder(MethodSorters.NAME_ASCENDING)** ustalono alfabetyczną kolejność wykonania metod testowych: ***testAddBook, testEquals, testGetBooksModel, testSearchBook, testSearchFreeBook***. Kolejna metoda testowa korzysta z danych utworzonych w poprzedniej metodzie testowej.

Zastosowanie adnotacji
@Test
@BeforeClass
@Before
@Parameter, @Parameters
@RunWith
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
@Category

Zastosowanie metod
static public void assertEquals(long expected, long actual) //k1.5
static public void assertTrue(boolean condition) // k1.1
static public void assertFalse(boolean condition) // k1.2
static public void assertEquals(Object expected, Object actual)//k1.3
static public void assertNotEquals(Object expected, Object actual)//k1.4

```
package subbusinesstier.entities;
import categories.Test_Entity;
import java.util.Arrays;
import java.util.Collection;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.BeforeClass;
import org.junit.FixMethodOrder;
import org.junit.experimental.categories.Category;
import org.junit.runner.RunWith;
import org.junit.runners.MethodSorters;
import org.junit.runners.Parameterized;
import testdata.Data;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
@Category({Test_Entity.class})
@RunWith(Parameterized.class)
public class TitleBookTest {

   static Data data;
   static TitleBook title;

   @Parameterized.Parameter
   public int number1;

   @Parameterized.Parameters
   public static Collection<Object[]> data() {
      Object[][] data1 = new Object[][]{{0}, {1}, {2}, {3}, {4}, {5}};
      return Arrays.asList(data1); }
```

```
@BeforeClass
   public static void setUpClass() {
      data = new Data();
   }

   @Before
   public void setUp() {
      title = data.titles[number1];
   }

   @Test
   public void testGetBooksModel() {
      System.out.println("getBooksModel");
      if (number1 == 0) {
         assertEquals(title.getBooksModel(), Arrays.asList(data.databooks1));                    //k1.3
         assertNotEquals(title.getBooksModel(), Arrays.asList(data.databooks3));                 //k1.4
      } else if (number1 == 5) {
         assertEquals(title.getBooksModel(), Arrays.asList(data.databooks2));            //k1.3
      }
   }

   @Test
   public void testEquals() {
      System.out.println("equals");
      for (int j = number1; j < 6; j++) {
         if (number1 == j) {
            assertTrue(data.titles[number1].equals(data.titles[j])); //k1.1 –test porównania
         } else                                                      // równych tytulow
            assertFalse(data.titles[number1].equals(data.titles[j])); //k1.2–test porównania
       }                                                            //różnych tytulow
      }

   @Test
   public void testAddBook() {
      System.out.println("addBook");
      for (int i = 0; i < data.bookdata.length; i++) {
         title.addBook(data.bookdata[i]);
         assertEquals(data.books[i], title.getBooks().get(i));        }          //k1.3
      for (int i = 0; i < data.bookdata.length; i++) {
         data.titles[0].addBook(data.bookdata[i]);
         assertEquals(title.getBooks().size(), data.bookdata.length); }    // k1.5- badanie spójnosci danych
  }

   @Test
   public void testSearchBook() {
      System.out.println("searchBook");
      for (int i = 0; i < data.books.length; i++) {
         Book book = title.searchBook(data.books[i]);
         assertEquals(book, data.books[i]);                                    //k1.3
      }
   }

@Test
   public void testSearchFreeBook() {
      System.out.println("searchFreeBook");
      for (int i = 0; i < data.bookdata.length; i++)
         title.getBooks().get(i).setReservations(data.reservationList);
      boolean result = title.searchFreeBook(data.dates[2]);
      assertTrue(result);
      result = title.searchFreeBook(data.dates[1]);
      assertFalse(result);
   }
}
```

2.6. Testy jednostkowe klasy *Facade* (wynik działania: p.2.7.1, 2.7.4) opierają się na wywołaniu trzech metod testowych, działających w kolejności alfabetycznej dzięki zastosowaniu adnotacji **@FixMethodOrder(MethodSorters.NAME_ASCENDING)**: ***test1AddTitleBook, test2AddBook, testAddClient, testAddReservation***. Przed wywołaniem metod testowych wywołana jest metoda ***SetUp*** dzięki zastosowaniu adnotacji **@BeforeClass** – metoda ta tworzy obiekt typu ***Facade*** *oraz* ***Data*** *z danymi wzorcowymi*. Metody działające w podanym porządku umożliwiają po dodaniu tytułów książek w metodzie ***test1AddTitleBook*** wykonać testy: dodawania książek w metodzie ***test2AddBook***, dodawanie klientów w metodzie testowej ***testAddClient*** oraz dodawanie rezerwacji w metodzie testowej ***testAddReservation***. Metoda testowa ***test1AddTitleBook*** testuje również przypadek podania niepoprawnej wartości w danych wejściowych, które powodują generowanie wyjątku przez metodę ***createTitleBook*** klasy ***Factory*** – wyjątek ten jest obsługiwany w metodzie testowej za pomocą mechanizmu **@Rule.** Za pomocą adnotacji **@Category** dokonano różnych klasyfikacji wszystkich metod testowych i wybranej metody ***testAddReservation***. **Należy wykonać interfejs pusty o nazwie Test_Reservation w celu zastosowania adnotacji @Category. Jest wstęp do utworzenia zestawu testów.**

Zastosowanie adnotacji
@Test
@BeforeClass
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
@Category
@Rule

Zastosowanie metod
static public void assertEquals(Object expected, Object actual)// k1.1
static public void assertEquals(long expected, long actual) //k1.2

```
package subbusinesstier;

import categories.Test_Control;
import categories.Test_Reservation;
import java.util.IllegalFormatCodePointException;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.FixMethodOrder;
import org.junit.Rule;
import org.junit.experimental.categories.Category;
import org.junit.rules.ExpectedException;
import org.junit.runners.MethodSorters;
import static subbusinesstier.FacadeTest.instance;
import subbusinesstier.entities.Book;
import subbusinesstier.entities.Client;
import subbusinesstier.entities.Reservation;
import subbusinesstier.entities.TitleBook;
import testdata.Data;

@Category({Test_Control.class}) //określenie kategorii testu, zastosowanie - p.2.7.1, 2.7.3
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class FacadeTest {

   static Data data;
   static Facade instance;
```

```
@Rule
public ExpectedException exception = ExpectedException.none();

@BeforeClass
public static void setUpClass() {
   instance = new Facade();
   data = new Data();
}

@Test
public void test1AddTitleBook() {
   System.out.println("addTitleBook");
   for (int i = 0; i < 6; i++) {
      instance.addTitleBook(data.titledata[i]);
      int number1 = instance.getTitleBooks().size();
      instance.addTitleBook(data.titledata[i]);                    //powtórzenia wartości elementów
      int number2 = instance.getTitleBooks().size();
      TitleBook result = instance.getTitleBooks().get(number2 - 1);
      assertEquals(data.titles[number2 - 1], result);              //k1.1 test dodawania tytulów
      assertEquals(number1, number2);          //k1.2 – test spójności danych podczas dodawanie tytulów
   }
   exception.expect(IllegalFormatCodePointException.class);        //obsługa wyjątku w testowanej metodzie
   exception.expectMessage("Code point = 0x0");
   instance.addTitleBook(data.titledata[12]);
}

 @Test
public void test2AddBook() {
   System.out.println("addBook");
   for (int i = 6; i < 12; i++) {
      for (int j = 0; j < 2; j++) {
         instance.addBook(data.titledata[i], data.bookdata[j]);
         int number1 = instance.getTitleBooks().get(i - 6).getBooks().size();
         instance.addBook(data.titledata[i], data.bookdata[j]);          //powtórzenia wartości elementów
         int number2 = instance.getTitleBooks().get(i - 6).getBooks().size();
         Book result = instance.getTitleBooks().get(i - 6).getBooks().get(number2 - 1);
         assertEquals(data.books[number2 - 1], result);               //k1.1 test dodawania ksiazek
         assertEquals(number1, number2);      //k1.2 – test spójności danych podczas dodawanie ksiazek
      }
   }
}

@Test
public void testAddClient() {
   System.out.println("addClient");
   for (int i = 0; i < 2; i++) {
      instance.addClient(data.clientdata[i]);
      int number1 = instance.getClients().size();
      instance.addClient(data.clientdata[i]);          //powtórzenia wartości elementów
      int number2 = instance.getClients().size();
      Client result = instance.getClients().get(number2 - 1);
      assertEquals(data.clients[number2 - 1], result);  //k1.1 test dodawania klientów
      assertEquals(number1, number2);      //k1.2 – test spójności danych podczas dodawanie klientów
   }
}
```

```
    @Test
    @Category(Test_Reservation.class)  //określenie kategorii testu – przykład zastosowania w p.2.7.4
    public void testAddReservation() {
        System.out.println("addReservation");
        int k = 1, i = 6, j = 2;
        instance.addReservation(data.titledata[i], data.clientdata[j], data.dates[0]);
        instance.addReservation(data.titledata[i], data.clientdata[j], data.dates[0]);
        int number2 = instance.getClients().get(j - 2).getReservations().size();
        instance.addReservation(data.titledata[i], data.clientdata[j], data.dates[0]);
                                                                    //powtórzenia wartości elementów
        int number3 = instance.getClients().get(j - 2).getReservations().size();
        Reservation result = instance.getClients().get(j - 2).getReservations().get(number3 - 1);
        assertEquals(data.reservations[k], result);     //k1.1 test dodawania rezerwacji
        assertEquals(number3, number2);                 //k1.2 – test spójności danych podczas dodawania rezerwacji
    }
}
```

2.7. Tworzenie zestawów testów

2.7.1. Wyniki testów wykonanych przez klasy należące również do kategorii @Category(Test_Control.class): *FactoryTest, FacadeTest*

```
package testsuite;
import categories.Test_Control;
import org.junit.experimental.categories.Categories;
import org.junit.runner.RunWith;
import subbusinesstier.FacadeTest;
import subbusinesstier.FactoryTest;
import subbusinesstier.entities.BookTest;
import subbusinesstier.entities.ClientTest;
import subbusinesstier.entities.ReservationTest;
import subbusinesstier.entities.TitleBookTest;

@Categories.SuiteClasses({FactoryTest.class, ReservationTest.class, BookTest.class, ClientTest.class,
TitleBookTest.class, FacadeTest.class})
@RunWith(Categories.class)
@Categories.IncludeCategory(Test_Control.class)
public class Suite11 {  }
```

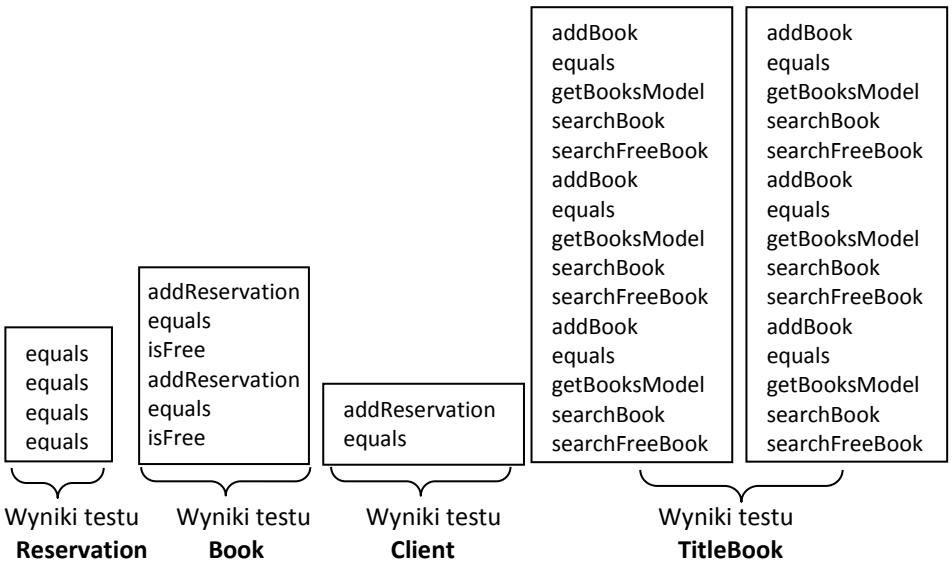
```
createTitleBook
createBook
createReservation
createClient
addTitleBook
addBook
addClient
addReservation
```

Wynik testu: ***FactoryTest, FacadeTest***

2.7.2. Wyniki testów wykonanych przez klasy należące tylko do kategorii @Category(Test_Entity.class): TitleBookTest.class, BookTest.class, ClientTest.class, ReservationTest.class – wariant 1

```
package testsuite;
import categories.Test_Control;
import org.junit.experimental.categories.Categories;
import org.junit.runner.RunWith;
import subbusinesstier.FacadeTest;
import subbusinesstier.FactoryTest;
import subbusinesstier.entities.BookTest;
import subbusinesstier.entities.ClientTest;
import subbusinesstier.entities.ReservationTest;
import subbusinesstier.entities.TitleBookTest;

@Categories.SuiteClasses({FactoryTest.class, ReservationTest.class, BookTest.class, ClientTest.class,
TitleBookTest.class, FacadeTest.class})
@RunWith(Categories.class)
@Categories.ExcludeCategory(Test_Control.class)
public class Suite12 { }
```



2.7.3. Wyniki testów wykonanych przez klasy należące do kategorii @Category(Test_Control.class) z wyłączeniem metody *testAddReservation* klasy *FacadeTest* zaliczonej do kategorii @Categorii(Test_Reservation.class) *: FactoryTest, FacadeTest*

```
package testsuite;
import categories.Test_Control;
import categories.Test_Reservation;
import org.junit.experimental.categories.Categories;
import org.junit.runner.RunWith;
import subbusinesstier.FacadeTest;
import subbusinesstier.FactoryTest;
import subbusinesstier.entities.BookTest;
import subbusinesstier.entities.ClientTest;
import subbusinesstier.entities.ReservationTest;
import subbusinesstier.entities.TitleBookTest;

@Categories.SuiteClasses({FactoryTest.class, ReservationTest.class, BookTest.class, ClientTest.class,
TitleBookTest.class, FacadeTest.class})
@RunWith(Categories.class)
@Categories.IncludeCategory(Test_Control.class)
@Categories.ExcludeCategory(Test_Reservation.class)
public class Suite13 {}
```

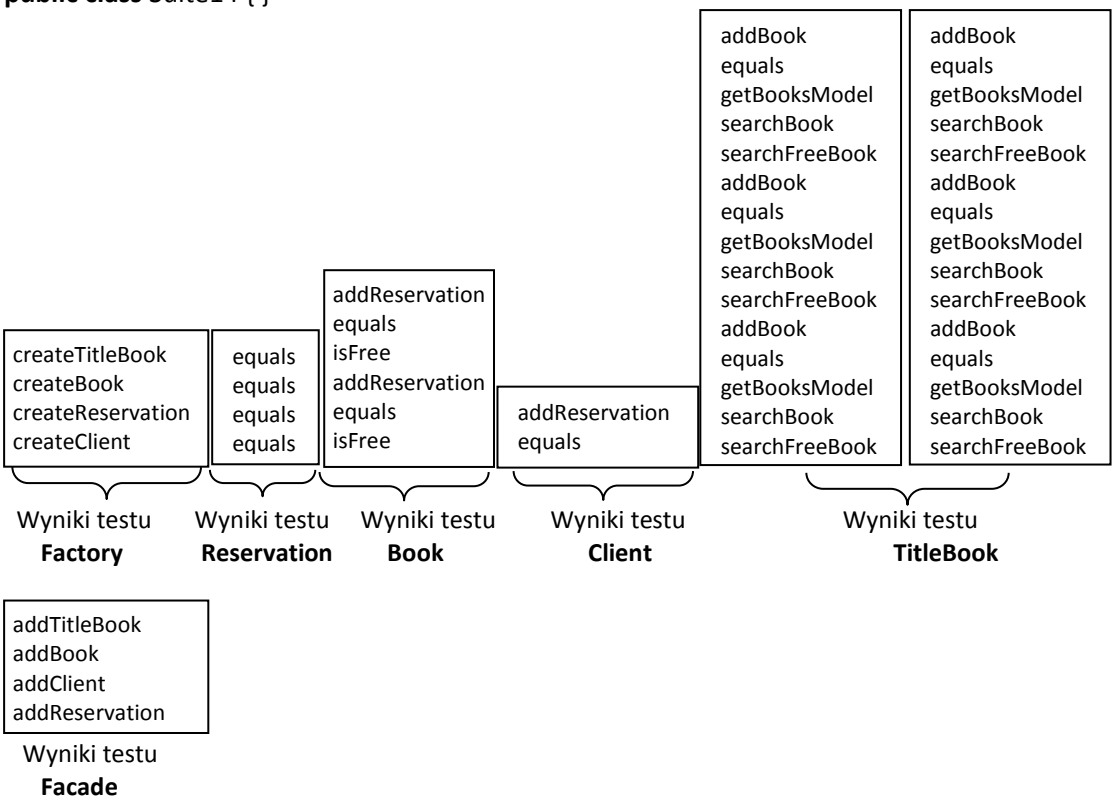
```
createTitleBook
createBook
createReservation
createClient
addTitleBook
addBook
addClient
```

Wynik testu: *FactoryTest, FacadeTest* z wyłączeniem metody testowej ***testAddReservation***

2.7.4. Wyniki testów wykonanych przez wszystkie klasy testujące – niezależnie od przypisanych kategorii

```
package testsuite;
import org.junit.experimental.categories.Categories;
import org.junit.runner.RunWith;
import subbusinesstier.FacadeTest;
import subbusinesstier.FactoryTest;
import subbusinesstier.entities.BookTest;
import subbusinesstier.entities.ClientTest;
import subbusinesstier.entities.ReservationTest;
import subbusinesstier.entities.TitleBookTest;

@Categories.SuiteClasses({FactoryTest.class, ReservationTest.class, BookTest.class, ClientTest.class,
TitleBookTest.class, FacadeTest.class})
@RunWith(Categories.class)
public class Suite14 { }
```



2.7.5. Wyniki testów wykonanych przez klasy należące tylko do kategorii @Category(Test_Entity.class): TitleBookTest.class, BookTest.class, ClientTest.class, ReservationTest.class – wariant 2

```
package testsuite;

import categories.Test_Entity;
import org.junit.experimental.categories.Categories;
import org.junit.runner.RunWith;
import subbusinesstier.FacadeTest;
import subbusinesstier.FactoryTest;
import subbusinesstier.entities.BookTest;
import subbusinesstier.entities.ClientTest;
import subbusinesstier.entities.ReservationTest;
import subbusinesstier.entities.TitleBookTest;


@Categories.SuiteClasses({FactoryTest.class, ReservationTest.class, BookTest.class, ClientTest.class,
TitleBookTest.class, FacadeTest.class})
@RunWith(Categories.class)
@Categories.IncludeCategory(Test_Entity.class)
public class Suite15 { }
```