Plan

Русский текст смотри ниже

# Plan of lesson

1. Homework explanation
2. Practice

# План на урок

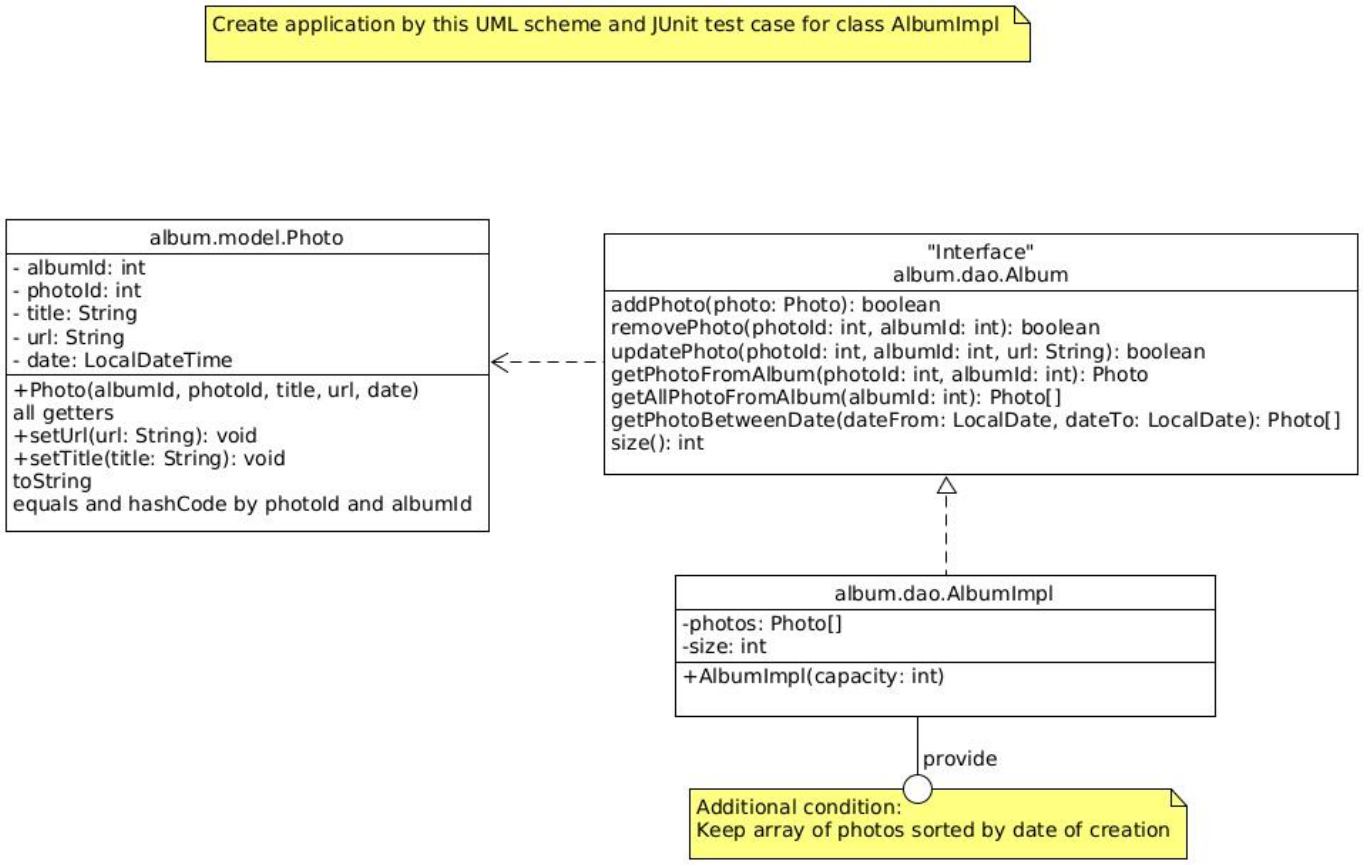
1. Разбор домашнего задания
2. Практическая работа

Theory

Русский текст смотри ниже

1. Create application Album by UML

1. Создать аппликацию Album согласно UML




Homework

Русский текст смотри ниже

**Task 1**

**Задача 1**

Перепишете приложение Album так что бы в массиве хранились сортированные фотографии.

- Фотографии должны быть отсортированные по двум полям
  - albumId
  - photoId

Пример: Дано:

| **title** | **Photo 1** | **Photo 2** | **Photo 3** | **Photo 4** | **Photo 5** | **Photo 6** | **Photo 7** |
|---|---|---|---|---|---|---|---|
| **albumId** | 2 | 1 | 2 | 2 | 1 | 3 | 10 |
| **photoId** | 2 | 1 | 5 | 3 | 3 | 1 | 1 |

Вот как выглядит таблица после сортировки:

| **title** | **Photo 2** | **Photo 5** | **Photo 1** | **Photo 3** | **Photo 4** | **Photo 6** | **Photo 7** |
|---|---|---|---|---|---|---|---|
| **albumId** | 1 | 1 | 2 | 2 | 2 | 3 | 10 |
| **photoId** | 1 | 3 | 2 | 5 | 3 | 1 | 1 |

Как видно из таблицы:

```
- Фотографии альбома с `albumId` 1 идут первыми, отсортированные по `photoId` внутри альбома (Photo 2 и Photo 5).
- Следующие идут фотографии альбома с `albumId` 2, также отсортированные по `photoId` (Photo 1, Photo 4, Photo 3). Заметьте, ч
- Фотография альбома с `albumId` 3 идёт следующей, так как у неё уникальный `albumId` (Photo 6).
- И, наконец, фотография с `albumId` 10, предполагаемая Photo 7, идёт последней.
```

- реализуйте все методы интерфейса Album, учтите что массив всегда должен оставаться сортированным.

Code

code/HW_Solution37/src/tasks/Main.java

```java
package tasks;

// Обратный массив: Напишите программу, которая создает новый массив,
// содержащий элементы исходного массива в обратном порядке, используя System.arraycopy().

// Слияние массивов: Напишите метод, который принимает два массива целых чисел и возвращает новый массив,
// который является результатом их слияния. Используйте System.arraycopy().

// Удаление дубликатов: Напишите программу, которая удаляет все дубликаты из отсортированного массива.
// Снова используйте System.arraycopy() для сдвига элементов.

import java.util.Arrays;


public class Main {


    public static void main(String[] args) {
        int[] reversed = reverse();
        System.out.println(Arrays.toString(reversed));

        int[] source1 = {1, 2, 3};
        int[] source2 = {10, 20, 30};

        int[] merged = merge(source1, source2);
        System.out.println(Arrays.toString(merged));

        System.out.println("\nFind Median");
        int[] array1 = {5, 17, 3, 9, 14, 2};
        int[] array2 = {5, 2, 18, 8, 3};
```

```
        double medianResult1 = findMedian(array1);
        double medianResult2 = findMedian(array2);

        System.out.println("Median Result1: " + medianResult1);
        System.out.println("Median Result2: " + medianResult2);

        System.out.println("\nRemove Duplicates");
        int[] arrayD1 = {1, 2, 2, 5, 6, 5, 1, 2, 1, 4, 2, 2, 2 };
        int[] resultRD = removeDuplicates(arrayD1);
        System.out.println(Arrays.toString(resultRD));

//        int[] resultRD2 = arrayWithoutDuplicate(arrayD1);
//        System.out.println(Arrays.toString(resultRD2));

    }

    public static int[] reverse() {
        int[] source = {1, 2, 3, 4, 5};
        int[] result = new int[source.length];

        for (int i = 0; i < source.length; i++) {
            System.arraycopy(source, i, result, source.length - 1 - i, 1);
        }
        return result;
    }

    public static int[] merge(int[] source1, int[] source2) {
        int[] result = new int[source1.length + source2.length];

        System.arraycopy(source1, 0, result, 0, source1.length);
        System.arraycopy(source2, 0, result, source1.length, source2.length);

        return result;// {1, 2, 3, 10, 20, 30}
    }

    // Создайте отсортированный массив случайных чисел. Напишите программу, которая находит медиану этого мас
    // Медианой ряда чисел (или медианой числового ряда) называется число, стоящее посередине упорядоченного
    // по возрастанию ряда чисел – в случае, если количество чисел нечётное. Если же количество чисел в ряду
    // то медианой ряда является полусумма двух стоящих посередине чисел упорядоченного по возрастанию ряда.
    //
    //Пример 1. Найти медиану числового ряда 5, 17, 3, 9, 14, 2.
    //
    //Решение. Записываем все числа ряда в порядке возрастания: 2, 3, 5, 9, 14, 17. Количество чисел в ряду ч
    // поэтому медиана этого ряда будет равна полусумме двух средних чисел: (5 + 9) / 2 = 7.
    //Пример 2. Найти медиану числового ряда 5, 2, 18, 8, 3.
    //
    //Решение. записываем все числа ряда в порядке возрастания: 2, 3, 5, 8, 18. Количество чисел в ряду нечёт
    // поэтому медиана этого ряда будет равна стоящему посередине числу, то есть равна 5.
    public static double findMedian(int[] source) {
//        Random random = new Random();
//        int[] randomNumbs = new int[6];
//        for (int i = 0; i < randomNumbs.length; i++) {
//            randomNumbs[i] = random.nextInt(15); // from 0 to 14
//        }

        Arrays.sort(source);
        double median;

        if (source.length % 2 == 0) {
            int mid1 = source[source.length / 2 - 1];
            int mid2 = source[source.length / 2];
```

```
            median = (double) (mid1 + mid2) / 2;
        } else {
            median = source[source.length / 2];
        }

        return median;
    }


    // Удаление дубликатов: Напишите программу, которая удаляет все дубликаты из отсортированного массива.
    // Снова используйте System.arraycopy() для сдвига элементов.
    public static int[] removeDuplicates(int[] source) {
        Arrays.sort(source);

        int j = 0;

        for (int i = 0; i < source.length - 1; i++) {
            if (source[i] != source[i + 1]) {
                source[j++] = source[i];
            }
        }

        source[j] = source[source.length - 1];

        int[] result = new int[j + 1];
        System.arraycopy(source, 0, result, 0, j + 1);

        return result;
    }


//    //todo fixit
//    public static int[] arrayWithoutDuplicate(int[] array) {
//        Arrays.sort(array);
//        for (int i = array.length - 1; i > 0; i--) {
//            if (array[i] == array[i - 1]) {
//                int[] result = new int [array.length - 1];
//                System.arraycopy(array, 0, result, 0, i);
//                System.arraycopy(array, i, result, i - 1, array.length - 1);
//                array = Arrays.copyOf(result, result.length);
//            }
//        }
//        return array;
//    }
}
```

code/Album/src/ait/album/dao/Album.java

```
package ait.album.dao.ait.album.dao;

import ait.album.dao.ait.album.model.Photo;

import java.time.LocalDate;

public interface Album {
    boolean addPhoto(Photo photo);

    boolean removePhoto(int photoId, int albumId);

    boolean updatePhoto(int photoId, int albumId, String url);
```

```
    Photo getPhotoFromAlbum(int photoId, int albumId);

    Photo[] getAllPhotoFromAlbum(int albumId);

    Photo[] getPhotoBetweenDate(LocalDate dateFrom, LocalDate dateTo);

    int size();
}
```

code/Album/src/ait/album/dao/AlbumImpl.java

```
package ait.album.dao.ait.album.dao;

import ait.album.dao.ait.album.model.Photo;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Arrays;

public class AlbumImpl implements Album {
    private Photo[] photos;
    private int size;

    public AlbumImpl(int capacity) {
        this.photos = new Photo[capacity];
    }


    @Override
    public boolean addPhoto(Photo photo) {

        boolean isInvalid = photo == null;
        if (isInvalid) {
            return false;
        }

        boolean isPhotoAlreadyAdded = getPhotoFromAlbum(photo.getPhotoId(), photo.getAlbumId()) != null;
        if (isPhotoAlreadyAdded) {
            return false;
        }

        boolean hasSlot = size == photos.length;
        if (hasSlot) {
            return false;
        }

        photos[size++] = photo;
        if (size > 0) {
            Arrays.sort(photos, 0, size);
        }
        return true;
    }

    @Override
    public boolean removePhoto(int photoId, int albumId) {
        for (int i = 0; i < size; i++) {
            if (photos[i].getAlbumId() == albumId && photos[i].getPhotoId() == photoId) {
                System.arraycopy(photos, i + 1, photos, i, size - i - 1);
                photos[size] = null;
                size--;
```

```
//                photos[i] = photos[size];
//                photos[size] = null;
//                size--;
                return true;
            }
        }

        return false;
    }

    @Override
    public boolean updatePhoto(int photoId, int albumId, String url) {
        Photo photoFromAlbum = getPhotoFromAlbum(photoId, albumId);

        if (photoFromAlbum != null) {
            photoFromAlbum.setUrl(url);
            return true;
        }

        return false;
    }

    @Override
    public Photo getPhotoFromAlbum(int photoId, int albumId) {
        for (int i = 0; i < size; i++) {
            if (photos[i].getAlbumId() == albumId && photos[i].getPhotoId() == photoId) {
                return photos[i];
            }
        }
        return null;
    }

    @Override
    public Photo[] getAllPhotoFromAlbum(int albumId) {
        Photo[] result = new Photo[size];
        int counter = 0;
        for (int i = 0; i < size; i++) {
            if (photos[i].getAlbumId() == albumId) {
                result[counter++] = photos[i];
            }
        }

        return Arrays.copyOf(result, counter);
    }

    @Override
    public Photo[] getPhotoBetweenDate(LocalDate dateFrom, LocalDate dateTo) {

        LocalDateTime dateFromPlusTime = dateFrom.atStartOfDay();
        LocalDateTime dateToPlusTime = dateTo.plusDays(1).atStartOfDay();

        Photo[] arrayPhotoBetweenDate = new Photo[size];

        int counter = 0;
        for (int i = 0; i < size; i++) {
            if (photos[i].getDate().isAfter(dateFromPlusTime) && photos[i].getDate().isBefore(dateToPlusTime
                arrayPhotoBetweenDate[counter++] = photos[i];
            }
        }

        return Arrays.copyOf(arrayPhotoBetweenDate, counter);
```

```
        }

        @Override
        public int size() {
            return size;
        }
    }
```

code/Album/src/ait/album/model/Photo.java

```
    package ait.album.dao.ait.album.model;

    import java.time.LocalDateTime;
    import java.util.Objects;

    public class Photo implements Comparable<Photo> {
        private int albumId;
        private int photoId;
        private String title;
        private String url;
        private LocalDateTime date;

        public Photo(int albumId, int photoId, String title, String url, LocalDateTime date) {
            this.albumId = albumId;
            this.photoId = photoId;
            this.title = title;
            this.url = url;
            this.date = date;
        }




        public int getAlbumId() {
            return albumId;
        }

        public int getPhotoId() {
            return photoId;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        public void setUrl(String url) {
            this.url = url;
        }

        public String getTitle() {
            return title;
        }

        public String getUrl() {
            return url;
        }

        public LocalDateTime getDate() {
            return date;
        }
```

```
    @Override
    public int compareTo(Photo other) {
        // int albumCompare = Integer.compare(this.getAlbumId(), other.albumId);
        // return albumCompare != 0 ? albumCompare : Integer.compare(this.getPhotoId(), other.getPhotoId());
        return this.getAlbumId() - other.getAlbumId();

    }

    @Override
    public String toString() {
        return "Photo albumId = " + albumId + ", photoId=" + photoId +
                ", title='" + title + '\'' +
                ", url='" + url + '\'' +
                ", date=" + date;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Photo photo = (Photo) o;
        return albumId == photo.albumId && photoId == photo.photoId;
    }

    @Override
    public int hashCode() {
        return Objects.hash(albumId, photoId);
    }
}
```

code/Album/src/ait/album/test/AlbumTest.java

```
package ait.album.dao.ait.album.test;

import ait.album.dao.ait.album.dao.Album;
import ait.album.dao.ait.album.dao.AlbumImpl;
import ait.album.dao.ait.album.model.Photo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Arrays;
import java.util.Comparator;

import static org.junit.jupiter.api.Assertions.*;

class AlbumTest {
    Album album;
    Photo[] ph;
    LocalDateTime now = LocalDateTime.now();
    Comparator<Photo> comparator = (p1, p2) -> {
        int res = Integer.compare(p1.getPhotoId(), p2.getPhotoId());
        return res != 0 ? res : Integer.compare(p1.getAlbumId(), p2.getAlbumId());
    };

    @BeforeEach
    void setUp() {
        album = new AlbumImpl(7);
```

```
        ph = new Photo[6];
        ph[0] = new Photo(1, 1, "title1", "url1", now.minusDays(7));
        ph[1] = new Photo(1, 2, "title2", "url2", now.minusDays(7));
        ph[2] = new Photo(1, 3, "title3", "url3", now.minusDays(5));
        ph[3] = new Photo(2, 1, "title1", "url1", now.minusDays(7));
        ph[4] = new Photo(2, 4, "title4", "url4", now.minusDays(2));
        ph[5] = new Photo(1, 4, "title4", "url1", now.minusDays(2));
        for (int i = 0; i < ph.length; i++) {
            album.addPhoto(ph[i]);
        }
    }

    @Test
    void addPhoto() {
        assertFalse(album.addPhoto(null));
        assertFalse(album.addPhoto(ph[1]));
        Photo photo = new Photo(1, 5, "title5", "url5", now.minusDays(3));
        assertTrue(album.addPhoto(photo)); // 7
        assertEquals(7, album.size());
        photo = new Photo(1, 6, "title6", "url6", now.minusDays(3));
        boolean isAdded = album.addPhoto(photo);
        assertFalse(isAdded);
    }

    @Test
    void removePhoto() {
        assertFalse(album.removePhoto(5, 1));
        assertTrue(album.removePhoto(1, 1));
        assertEquals(5, album.size());
        assertNull(album.getPhotoFromAlbum(1, 1));
    }

    @Test
    void updatePhoto() {
        assertTrue(album.updatePhoto(1, 1, "newUrl"));
        assertEquals("newUrl", album.getPhotoFromAlbum(1, 1).getUrl());
    }

    @Test
    void getPhotoFromAlbum() {
        assertEquals(ph[0], album.getPhotoFromAlbum(1, 1));
        assertNull(album.getPhotoFromAlbum(1, 5));
    }

    @Test
    void getAllPhotoFromAlbum() {
        Photo[] actual = album.getAllPhotoFromAlbum(2); //{ph[4], ph[3]}
        Photo[] expected = {ph[3], ph[4]};
        Arrays.sort(actual, comparator);
        assertArrayEquals(expected, actual);
    }

    @Test
    void getPhotoBetweenDate() {
        LocalDate ld = LocalDate.now();
        Photo[] actual = album.getPhotoBetweenDate(ld.minusDays(5), ld.minusDays(2));
        Photo[] expected = {ph[2], ph[5], ph[4]};
        Arrays.sort(actual, comparator);
        assertArrayEquals(expected, actual);
    }
```

```
    @Test
    void size() {
        assertEquals(6, album.size());
    }
}
```

code/SortedAlbum/src/dao/Album.java

```
package dao;


import model.Photo;

import java.time.LocalDate;

public interface Album {
    boolean addPhoto(Photo photo);

    boolean removePhoto(int photoId, int albumId);

    boolean updatePhoto(int photoId, int albumId, String url);

    Photo getPhotoFromAlbum(int photoId, int albumId);

    Photo[] getAllPhotoFromAlbum(int albumId);

    Photo[] getPhotoBetweenDate(LocalDate dateFrom, LocalDate dateTo);

    int size();
}
```

code/SortedAlbum/src/dao/AlbumImpl.java

```
package dao;


import model.Photo;

import java.time.LocalDate;

public class AlbumImpl implements Album {
    private Photo[] photos;
    private int size;

    public AlbumImpl(int capacity) {
        this.photos = new Photo[capacity];
    }

    @Override
    public boolean addPhoto(Photo photo) {

        return false;
    }

    @Override
    public boolean removePhoto(int photoId, int albumId) {

        return false;
    }
```

```
    @Override
    public boolean updatePhoto(int photoId, int albumId, String url) {

        return false;
    }

    @Override
    public Photo getPhotoFromAlbum(int photoId, int albumId) {

        return null;
    }

    @Override
    public Photo[] getAllPhotoFromAlbum(int albumId) {

        return new Photo[0];
    }

    @Override
    public Photo[] getPhotoBetweenDate(LocalDate dateFrom, LocalDate dateTo) {

        return new Photo[0];
    }

    @Override
    public int size() {
        return size;
    }
}
```

code/SortedAlbum/src/model/Photo.java

```
package model;

import java.time.LocalDateTime;
import java.util.Objects;

// todo:  implements Comparable<Photo>
// сравнивать фотографии по albumId, photoId
public class Photo implements Comparable<Photo> {
    private int albumId;
    private int photoId;
    private String title;
    private String url;
    private LocalDateTime date;

    public Photo(int albumId, int photoId, String title, String url, LocalDateTime date) {
        this.albumId = albumId;
        this.photoId = photoId;
        this.title = title;
        this.url = url;
        this.date = date;
    }

    public int getAlbumId() {
        return albumId;
    }

    public int getPhotoId() {
        return photoId;
```

```
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public String getTitle() {
        return title;
    }

    public String getUrl() {
        return url;
    }

    public LocalDateTime getDate() {
        return date;
    }

    @Override
    public int compareTo(Photo other) {
        int albumCompare = Integer.compare(this.getAlbumId(), other.albumId);
        return albumCompare != 0 ? albumCompare : Integer.compare(this.getPhotoId(), other.getPhotoId());
    }

    @Override
    public String toString() {
        return "Photo albumId = " + albumId + ", photoId=" + photoId +
                ", title='" + title + '\'' +
                ", url='" + url + '\'' +
                ", date=" + date;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Photo photo = (Photo) o;
        return albumId == photo.albumId && photoId == photo.photoId;
    }

    @Override
    public int hashCode() {
        return Objects.hash(albumId, photoId);
    }
}
```

code/SortedAlbum/src/test/PhotoTest.java

```
package test;

import model.Photo;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import java.util.Arrays;
```

```
/**
 * @author Andrej Reutow
 * created on 06.11.2023
 */
class PhotoTest {

    private final Photo photo1_1 = new Photo(1, 1, null, null, null);
    private final Photo photo1_2 = new Photo(1, 2, null, null, null);
    private final Photo photo2_1 = new Photo(2, 1, null, null, null);

    @Test
    public void test_compare() {


        int resultCompare1 = photo1_1.compareTo(photo1_2);
        Assertions.assertTrue(resultCompare1 < 0);

        int resultCompare2 = photo1_1.compareTo(photo2_1);
        Assertions.assertTrue(resultCompare2 < 0);
    }


    @Test
    public void test_sortArray() {
        Photo[] photos = new Photo[3];
        photos[0] = photo2_1;
        photos[1] = photo1_1;
        photos[2] = photo1_2;

        Arrays.sort(photos);

        Assertions.assertArrayEquals(new Photo[]{photo1_1, photo1_2, photo2_1}, photos);
    }
}
```

code/SortedAlbum/src/test/AlbumTest.java

```
package test;


import dao.Album;
import dao.AlbumImpl;
import model.Photo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Arrays;
import java.util.Comparator;

import static org.junit.jupiter.api.Assertions.*;

class AlbumTest {
    Album album;
    Photo[] ph;
    LocalDateTime now = LocalDateTime.now();
    Comparator<Photo> comparator = (p1, p2) -> {
        int res = Integer.compare(p1.getPhotoId(), p2.getPhotoId());
        return res != 0 ? res : Integer.compare(p1.getAlbumId(), p2.getAlbumId());
```

```
};

@BeforeEach
void setUp() {
    album = new AlbumImpl(7);
    ph = new Photo[6];
    ph[0] = new Photo(1, 1, "title1", "url1", now.minusDays(7));
    ph[1] = new Photo(1, 2, "title2", "url2", now.minusDays(7));
    ph[2] = new Photo(1, 3, "title3", "url3", now.minusDays(5));
    ph[3] = new Photo(2, 1, "title1", "url1", now.minusDays(7));
    ph[4] = new Photo(2, 4, "title4", "url4", now.minusDays(2));
    ph[5] = new Photo(1, 4, "title4", "url1", now.minusDays(2));
    for (int i = 0; i < ph.length; i++) {
        album.addPhoto(ph[i]);
    }
}

@Test
void addPhoto() {
    assertFalse(album.addPhoto(null));
    assertFalse(album.addPhoto(ph[1]));
    Photo photo = new Photo(1, 5, "title5", "url5", now.minusDays(3));
    assertTrue(album.addPhoto(photo)); // 7
    assertEquals(7, album.size());
    photo = new Photo(1, 6, "title6", "url6", now.minusDays(3));
    boolean isAdded = album.addPhoto(photo);
    assertFalse(isAdded);
}

@Test
void removePhoto() {
    assertFalse(album.removePhoto(5, 1));
    assertTrue(album.removePhoto(1, 1));
    assertEquals(5, album.size());
    assertNull(album.getPhotoFromAlbum(1, 1));
}

@Test
void updatePhoto() {
    assertTrue(album.updatePhoto(1, 1, "newUrl"));
    assertEquals("newUrl", album.getPhotoFromAlbum(1, 1).getUrl());
}

@Test
void getPhotoFromAlbum() {
    assertEquals(ph[0], album.getPhotoFromAlbum(1, 1));
    assertNull(album.getPhotoFromAlbum(1, 5));
}

@Test
void getAllPhotoFromAlbum() {
    Photo[] actual = album.getAllPhotoFromAlbum(2); //{ph[4], ph[3]}
    Photo[] expected = {ph[3], ph[4]};
    Arrays.sort(actual, comparator);
    assertArrayEquals(expected, actual);
}

@Test
void getPhotoBetweenDate() {
    LocalDate ld = LocalDate.now();
    Photo[] actual = album.getPhotoBetweenDate(ld.minusDays(5), ld.minusDays(2));
```

```
        Photo[] expected = {ph[2], ph[5], ph[4]};
        Arrays.sort(actual, comparator);
        assertArrayEquals(expected, actual);
    }

    @Test
    void size() {
        assertEquals(6, album.size());
    }
}
```