Plan

Русский текст смотри ниже

Plan of lesson

1. Homework explanation
2. Java Time API

План на урок

1. Разбор домашнего задания
2. Java Time API

Theory

Русский текст смотри ниже

1. Java Time API is a set of classes and interfaces for working with date, time, calendar
2. The LocalDate, LocalTime, LocalDateTime classes are designed to create objects associated with date, time, date-time, respectively. Objects of these classes are immutable. These classes do not have public constructors. Objects of these classes are created through static methods. For example, the now() method creates an object associated with the current date or time. Using the of method, you can create an object associated with a specific date or time.
3. LocalDate, LocalTime, LocalDateTime objects have many getters that allow you to get information about a date or time. For example, the getDayOfYear method will return the ordinal number of the day in the year. And the getDayOfWeek method will return the enum DayOfWeek associated with the day of the week.
4. Objects LocalDate, LocalTime, LocalDateTime are comparable. In addition, using the isBefore and isAfter methods, you can find out whether the date comes before or after the argument.
5. The plus, minus methods and their variants allow you to add or subtract time intervals from dates. Using the enum ChronoUnit, you can determine which time periods we are talking about. For example, ChronoUnit.HOURS is hours, and ChronoUnit.MINUTES is minutes. And so on.
6. ChronoUnit objects have methods. For example ChronoUnit.YEARS.between() takes two dates and returns the total number of years between them.
7. To convert dates to String and back, you must define a format in accordance with ISO 8601. DateTimeFormatter is a class responsible for formats. It has predefined formats in the form of constants. Or, using the static DateTimeFormatter.ofPattern() method, you can create a format based on a pattern.

8. The parse() method for date and time objects can create an object from a string in accordance with the specified format. The format() method, on the contrary, converts an object into a string in accordance with the specified format.
9. Here is a link to the official Oracle tutorial on Java Time API and a link to DateTimeFormatter

1. Java Time API, это набор классов и интерфейсов для работы с датой, временем, календарем
2. Классы LocalDate, LocalTime, LocalDateTime предназначены для создания объектов связанных с датой, временем, датой-временем, соответсвенно. Объекты этих классов являются неизменными (immutable). У этих классов нет публичных конструкторов. Объекты этих классов создаются через статические методы. Например метод now() создает объект связанный с текущей датой или временем. При помощи метода of можно создать объект связанный с конкретной датой или временем.
3. У объектов LocalDate, LocalTime, LocalDateTime есть множество геттеров, позволяющих получить информацию о дате или времени. Например метод getDayOfYear вернет порядковый номер дня в году. А метод getDayOfWeek вернет enum DayOfWeek связанный с днем недели.
4. Объекты LocalDate, LocalTime, LocalDateTime сравниваемы (Comparable). Кроме того, при помощи методов isBefore, isAfter можно узнать дата идет до или после аргумента.
5. Методы plus, minus и их варианты посзовляют прибавлять или вычитать из дат временные промежутки. При помощи enum ChronoUnit можно определять о каких временных промежутках идет речь. Например ChronoUnit.HOURS - это часы, а ChronoUnit.MINUTES - минуты. И т. п.
6. У объектов ChronoUnit есть методы. Например ChronoUnit.YEARS.between() принимает две даты и возвращает полное колличество лет между ними.
7. Для преобразований дат в String и обратно необходимо определять формат в соответствии с ISO 8601. DateTimeFormatter это класс отвечающий за форматы. Он имеет предопределенные форматы в виде констант. Или при помощи статического метода DateTimeFormatter.ofPattern() можно создать формат по шаблону.
8. Метод parse() у объектов даты, времени может создать объект из стринга в соответствии с указанным форматом. Метод format() наоборот, преобразует объект в стринг в соответствии с заданным форматом.
9. Вот ссылка на официальный Oracle туториал по Java Time API и ссылка на DateTimeFormatter

Homework

Русский текст смотри ниже

Task 1

In the DateOperation class from class work, implement the getAge method, which accepts the date of birth as a string, and returning age.

Task 2

In the DateOperation class from class work, implement the sortStringDates method, which accepts an array of dates in the form an array of strings, and returning a sorted array of date strings.

To check, use the DateOperationTest class with unit tests.

Задача 1

В классе DateOperation из классной работы, реализовать метод getAge, принимающий дату рождения в виде стринга, и возвращающий возраст.

Задача 2

В классе DateOperation из классной работы, реализовать метод sortStringDates, принимающий массив дат в виде массива стрингов, и возвращающий отсортированный массив стрингов-дат.

Для проверки используйте класс DateOperationTest с юнит тестами.

Code

code/src/homeworks/CompanyImpl.java

```
package ait.employee.dao;

import ait.employee.model.Employee;
import ait.employee.model.SalesManager;

public class CompanyImpl implements Company {
    private Employee[] employees;
    private int size;

    public CompanyImpl(int capacity) {
        employees = new Employee[capacity];
    }

    @Override
    public boolean addEmployee(Employee employee) {
        //TODO throw RuntimeException if employee == null
        if(employee == null){
            throw new RuntimeException();
        }
        if (size == employees.length || findEmployee(employee.getId()) != n
            return false;
        }
//        employees[size] = employee;
//        size++;
        employees[size++] = employee;
        return true;
    }

    @Override
    public Employee removeEmployee(int id) {
        for (int i = 0; i < size; i++) {
            if (employees[i].getId() == id) {
                Employee victim = employees[i];
//                employees[i] = employees[size - 1];
//                employees[size - 1] = null;
//                size--;
                employees[i] = employees[--size];
                employees[size] = null;
                return victim;
            }
        }
        return null;
    }

    @Override
    public Employee findEmployee(int id) {
        for (int i = 0; i < size; i++) {
            if (employees[i].getId() == id) {
                return employees[i];
            }
        }
        return null;
    }

    @Override
    public double totalSalary() {
        double res = 0;
        for (int i = 0; i < size; i++) {
            res += employees[i].calcSalary();
        }
        return res;
    }

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

    @Override
    public double avgSalary() {
        return totalSalary() / size;
    }

    @Override
    public double totalSales() {
        double res = 0;
        for (int i = 0; i < size; i++) {
            if (employees[i] instanceof SalesManager) {
                SalesManager salesManager = (SalesManager) employees[i];
                res += salesManager.getSalesValue();
            }
        }
        return res;
    }

    @Override
    public void printEmployees() {
        for (int i = 0; i < size; i++) {
            System.out.println(employees[i]);
        }
    }

    @Override
    public Employee[] findEmployeesHoursGreaterThan(int hours) {
        int count = 0;
        for (int i = 0; i < size; i++) {
            if (employees[i].getHours() > hours) {
                count++;
            }
        }
        Employee[] res = new Employee[count];
        for (int i = 0, j = 0; j < res.length; i++) {
            if (employees[i].getHours() > hours) {
                res[j++] = employees[i];
//                    j++;
            }
        }
        return res;
    }

    @Override
    public Employee[] findEmployeesSalaryRange(int minSalary, int maxSalary
        int count = 0;
        for (int i = 0; i < size; i++) {
            if (employees[i].calcSalary() >= minSalary && employees[i].calc
                count++;
            }
        }
        Employee[] res = new Employee[count];
        for (int i = 0, j = 0; j < res.length; i++) {
            if (employees[i].calcSalary() >= minSalary && employees[i].calc
                res[j++] = employees[i];
            }
        }
        return res;
    }
}
```

code/src/homeworks/CompanyTest.java

```
package ait.employee.test;

import ait.employee.dao.Company;
import ait.employee.dao.CompanyImpl;
import ait.employee.model.Employee;
import ait.employee.model.Manager;
import ait.employee.model.SalesManager;
import ait.employee.model.WageEmployee;
import org.junit.jupiter.api.Test;

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

class CompanyTest {
    Company company;
    Employee[] firm;

    @org.junit.jupiter.api.BeforeEach
    void setUp() {
        company = new CompanyImpl(5);
        firm = new Employee[4];
        firm[0] = new Manager(1000, "John", "Smith", 160, 5000, 5);
        firm[1] = new WageEmployee(2000, "Ann", "Smith", 160, 15);
        firm[2] = new SalesManager(3000, "Peter", "Jackson", 160, 19000, 0.
        firm[3] = new SalesManager(4000, "Rabindranate", "Agraval", 80, 200
        for (int i = 0; i < firm.length; i++) {
            company.addEmployee(firm[i]);
        }
    }

    @org.junit.jupiter.api.Test
    void addEmployee() {
        //TODO assert exception if employee is null
        boolean flag;
        try {
            company.addEmployee(null);
            flag = true;
        } catch (RuntimeException e) {
            flag = false;
        }
        assertFalse(flag);
//          assertFalse(company.addEmployee(null));
        assertFalse(company.addEmployee(firm[1]));
        Employee employee = new SalesManager(5000, "Peter", "Jackson", 160,
        assertTrue(company.addEmployee(employee));
        assertEquals(5, company.quantity());
        employee = new SalesManager(6000, "Peter", "Jackson", 160, 19000, 0
        assertFalse(company.addEmployee(employee));
    }

    @org.junit.jupiter.api.Test
    void removeEmployee() {
        Employee employee = company.removeEmployee(3000);
        assertEquals(firm[2], employee);
        assertEquals(3, company.quantity());
        assertNull(company.removeEmployee(3000));
    }
    @org.junit.jupiter.api.Test
    void findEmployee() {
        assertEquals(firm[1], company.findEmployee(2000));
        assertNull(company.findEmployee(5000));
    }

    @org.junit.jupiter.api.Test
    void totalSalary() {
        assertEquals(12280, company.totalSalary(), 0.01);
    }

    @org.junit.jupiter.api.Test
    void quantity() {
        assertEquals(4, company.quantity());
    }

    @org.junit.jupiter.api.Test
    void avgSalary() {
        assertEquals(12280.0 / 4, company.avgSalary(), 0.01);
    }

    @org.junit.jupiter.api.Test
    void totalSales() {
        assertEquals(39000, company.totalSales(), 0.01);
    }

    @org.junit.jupiter.api.Test
    void printEmployees() {
        company.printEmployees();
    }

    @Test
    void findEmployeesHoursGreaterThan() {
        Employee[] actual = company.findEmployeesHoursGreaterThan(100);
        Employee[] expected = {firm[0], firm[1], firm[2]};
        assertArrayEquals(expected, actual);
    }

    @Test
    void findEmployeesSalaryRange() {
        Employee[] actual = company.findEmployeesSalaryRange(2000, 2400);
        Employee[] expected = {firm[2], firm[3]};
        assertArrayEquals(expected, actual);
    }

    @Test
    void testIncrement() {
        int a = 10;
        int b = a++ + ++a;
        assertEquals(12, a);
        assertEquals(22, b);
    }
}
```

code/src/TimeAppl.java

```
package ait.time;

/*
Joda Time
Java Time API
1) Current and other date and time (V)
2) plus and minus date and time units (V)
3) Compare of dates (V)
4) Date Format ISO-8601
5) Zoned date time
6) Period and Duration (V)
7) Customization

 */

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.Locale;

public class TimeAppl {
    public static void main(String[] args) {
        LocalDate localDate = LocalDate.now();
        System.out.println(localDate);
        LocalTime localTime = LocalTime.now();
        System.out.println(localTime);
        LocalDateTime localDateTime = LocalDateTime.now();
        System.out.println(localDateTime);
        LocalDate gagarin = LocalDate.of(1961, 4, 12);
        System.out.println(gagarin);
        System.out.println(gagarin.getDayOfMonth());
        System.out.println(gagarin.getDayOfWeek());
        System.out.println(gagarin.getDayOfYear());
        System.out.println(localDate.isAfter(gagarin));
        System.out.println(localDate.isBefore(gagarin));
        System.out.println(gagarin.isBefore(gagarin));
        System.out.println(gagarin.isAfter(gagarin));
        System.out.println(localDate.compareTo(gagarin));
        System.out.println(gagarin.compareTo(localDate));
        System.out.println(gagarin.compareTo(gagarin));
        LocalDate newDate = localDate.plusDays(10);
        System.out.println(newDate);
        newDate = localDate.plusWeeks(7);
        System.out.println(newDate);
        newDate = localDate.minusMonths(3);
        System.out.println(newDate);
        newDate = localDate.plus(14, ChronoUnit.DECADES);
        System.out.println(newDate);
        LocalTime newTime = localTime.plus(14, ChronoUnit.MINUTES);
        System.out.println(newTime);
        LocalDateTime newLocalDateTime = localDateTime.plus(9, ChronoUnit.H
        System.out.println(newLocalDateTime);
        long period = ChronoUnit.DAYS.between(gagarin, localDate);
        System.out.println(period);
        period = ChronoUnit.YEARS.between(gagarin, localDate);
        System.out.println(period);
        period = ChronoUnit.WEEKS.between(gagarin, localDate);
        System.out.println(period);
        period = ChronoUnit.MONTHS.between(localDate, gagarin);
        System.out.println(period);
        System.out.println("===== DateTimeFormatter =====");
        DateTimeFormatter df = DateTimeFormatter.BASIC_ISO_DATE;
        String date = gagarin.format(df);
        System.out.println(date);
        date = gagarin.toString();
        System.out.println(date);
        df = DateTimeFormatter.ISO_LOCAL_DATE;
        date = localDate.format(df);
        System.out.println(date);
        df = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        date = gagarin.format(df);
        System.out.println(date);
        df = DateTimeFormatter.ofPattern("dd/M/yyyy");
        date = gagarin.format(df);
        System.out.println(date);
        df = DateTimeFormatter.ofPattern("dd/MMM/yyyy");
        date = gagarin.format(df);
        System.out.println(date);
        df = DateTimeFormatter.ofPattern("dd/MMMM/yyyy");
        date = gagarin.format(df);
        System.out.println(date);
        df = DateTimeFormatter.ofPattern("dd/MMMM/yyyy", Locale.FRANCE);
        date = gagarin.format(df);
        System.out.println(date);
        df = DateTimeFormatter.ofPattern("dd/MMMM/yyyy", Locale.forLanguage
        date = gagarin.format(df);
        System.out.println(date);
        date = "31/10/2023";
        df = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        newDate = LocalDate.parse(date, df);
        System.out.println(newDate.toString());
        System.out.println(newDate.format(DateTimeFormatter.ofPattern("dd/M
    }
}
```

code/src/DateOperation.java

```
package ait.time.utils;

public class DateOperation {
    public static int getAge(String birthDate) {
        // TODO
        return 0;
    }

    public static String[] sortStringDates(String[] dates) {
        // TODO
        return null;
    }

}
```

code/src/DateOperationTest.java

```
package ait.time.test;
```

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

import ait.time.utils.DateOperation;
import org.junit.jupiter.api.Test;

class DateOperationTest {

    @Test
    void getAge() {
        assertEquals(62, DateOperation.getAge("12/04/1961"));
        assertEquals(61, DateOperation.getAge("1961-11-28"));
    }

    @Test
    void sortStringDates() {
        String[] dates = {"2000-12-01", "10/12/2000", "1970-08-12", "2010-1
        String[] actual = DateOperation.sortStringDates(dates);
        String[] expected = {"1970-08-12", "2000-12-01", "10/12/2000", "201
        assertArrayEquals(expected, actual);
    }
}
```

code/src/ExeptionAppl.java

```
404: Not Found
```

code/src/NoSolutionException.java

```
package ait.exception;

public class NoSolutionException extends Exception {
    public NoSolutionException(){}
    public NoSolutionException(String message){
        super(message);
    }
}
```

code/src/SolutionAnyNumberException.java

```
package ait.exception;

public class SolutionAnyNumberException extends RuntimeException {

    public SolutionAnyNumberException(){}
```

```
        public SolutionAnyNumberException(String message){
            super(message);
        }
    }
```