Informatik 2 Group 2

Lab 4 Abstract Data Types

By Akrem Cheniour (s0564828) & Bartholomäus Berresheim (s0568624)

Index

- Introduction
- Pre-lab
    1. How do Julian Dates work? Search the Internet for a...
    2. Think up some good test cases for testing a JulianDate...
- Assignments
    1. Implement the abstract data type Julian Date you specified…
    2. Now make a little program that uses your Julian Date class…
    3. A metric system is proposed to reform the calendar...
- Reflections
    - Akrem
    - Bartholomäus
- Code

Introduction

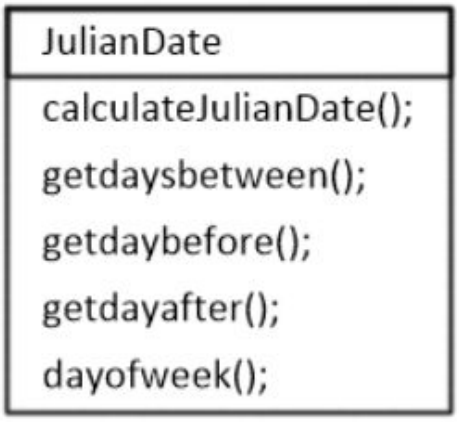
In this weeks Lab we worked on Julian Dates, the ways they can be used to calculate certain things like the period of days between 2 dates or how old someone is, in a metric calendar.

Pre-lab

1.How do Julian Dates work? Search the Internet for a description and put a link to the source in your report. What methods should a class that offers Julian Dates have? Write an abstract data type that expresses this.

We found the following description of how Julian Dates work:
"Julian day is the continuous count of days since the beginning of the Julian Period.[...] The Julian date (JD) of any instant is the Julian day number plus the fraction of a day since the preceding noon in Universal Time." - (https://en.wikipedia.org/wiki/Julian_day)

An Abstract Data Type expressing what methods a class that offers Julian Dates should have, looks like this:



2. Think up some good test cases for testing a JulianDate collection of classes. But what is a test case? A pair (input, expected output). That means, that you have to figure out before running the code what the program should output. Include some arithmetic methods such as daysBetween, tomorrow and yesterday.
We wrote one test case for each method.
calculateJulianDate(): (13.04.2013, 2456395)
getdaysbetween(): (12.09.2010,17.12.2011),461)
getdaybefore(): (06.02.823, 2021694)
getdayafter(): (10.05.-2816,693010)
dayofweek(): (06.06.666, Saturday)

Assignments

1. Implement the abstract data type Julian Date you specified in the prelab as a class. If you are missing any methods, explain in your report how you figured out that they were missing, and how you implemented them. Construct a test harness—another class that tests your ADT class and tries to find errors in your implementation.

To start things of, we implemented our abstract data type from P2 as an interface called JD.

```
public interface JD {

    public int calculateJD (int date);
    public int getdaysbetween (int date1, int date2);
    public String ndaysago (int date, int n);
    public String inNdays (int date, int n);
    public String dayoftheweek (int date);
    public String jultogreg (int date);

}
```

We then continued by creating another class, called JulianDate, in which we implemented the JD interface and proceeded to define the methods.

The first method calculateJD() had the purpose of converting the inputted gregorian date to a julian date. To do that, we first needed to decide where to put our 0 for the Julian date, we could have chosen to take the original January the first 4713 BC but without taking a premade algorithm, it seemed rather complicated taking the missing days and the irregularity of leap-years (comparison between before and after the reform 1582) into account. In the end we decided on January the first 1900, this decision was made in consideration of the 2nd assignment, in which we had to write a Birthday class. We wanted to write a program that works for every living person, and since the oldest person alive was born after 1903, we thought 1900 to be a good starting point.

After deciding that, we continued with the programming. In the beginning of the method, we separated the inputted date (yyyyMMdd format) into year, month and day, using modulo, division etc. (since the int rounds off the decimal points, we don't have to worry about getting unwanted results when using divisions).

```
day = date%100;
month = (date-day)%10000/100;
year = date/10000;
```

We choose that particular format, since year is the only variable that doesn't have a constant amount of digits.
This was followed by a while loop, using which we calculated the days passed in all the previous year (1900 to year-1), taking the leap years into consideration. We also had to subtract 1 to the JulianDate counter, since 01.01.1900 is our 0 and not 1.

```
year -= 1;
JulianDate = day - 1;
while (year >= 1900)
{
    if (year%4 ==0 && year%100 != 0 || year%400 == 0)
    {
        JulianDate +=366;
    }
    else {
        JulianDate += 365;
    }
    year--;
}
```

Previously, we had a similar way of checking the leap years, except that the leap-year-exceptions were limited to 1900 and 2100. But during a discussion between Bartholomäus and Juri, he showed us this way of

checking for exceptions and was nice enough to give us the permission to use it.
After the loop, we added the days passed in the previous months of the current year to JulianDate, by calling a predefined array with [month-1] as index, that contained these values.

```
private int [] DaysBeforeMonths = {0,31,59,90,120,151,181,212,243,273,304,334};

 JulianDate += DaysBeforeMonths[month-1];
```

and at the end of that, we set the year variable back to its original value and checked if the current year is a leap-year and if the month is at least March, in which case we would add another day to the JulianDate variable.

```
        year = date/10000;
        if ((year%4 ==0 && year%100 != 100 || year%400 == 0) && month>2)
        {
            JulianDate +=1;
            }

    return JulianDate;
}
```

At the end we return JulianDate.

Following that we defined the getdaysbetween() method. It has 2 dates as input, which it uses as parameters to call the calculateJD() method and then compares the returns in an if-else statement that subtracts the smaller one from the bigger one. The result we get represents the number of days between the imputed dates.

```
public int getdaysbetween(int date1, int date2) {
    if (calculateJD(date1) < calculateJD(date2))
    {
        daysbetween = calculateJD(date2) - calculateJD(date1);
    }
    else {
        daysbetween = calculateJD(date1) - calculateJD(date2);
    }
    return daysbetween;
}
```

After some deliberation about the tommorow() and yesterday() methods mentioned in the P2 task, we changed them into inNdays() and ndaysago(), which would allow a broader usage. And for them to work properly, we first had to write a method that converts julian to gregorian. That method was missing from our interface, so we added it quickly.
We called that method jultogreg(), in it we set year to 1900 and “day” to input+1 (or else the resulting day would be shifted by one). We then

created a while loop that would check if “day” is bigger than 366 or 365 and if that was the case, it would subtract 365 or 366 from “day” and add 1 to “year”.

```
@Override
public String jultogreg (int JulianDate)
{
    day = JulianDate+1;


    year = 1900;

    while (day > ((year%4 ==0 && year%100 != 0) || (year%400 == 0)?366:365))
    {
        day -= ((year%4 ==0 && year%100 != 0) || (year%400 == 0)?366:365);

        year++;
    }
```

We then used a for loop in which we had an if-statement that would look if “day” is bigger than the values in the previously mentioned array and if that is the case it would subtract the biggest of the array-values from “day”, set “month” to index+1 and break the loop.
To finish things of it would assemble day, month and year into a String. before outputting it.

```
for (int i =11; i >-1; i--)
{
    if (day >DaysBeforeMonths[i]) {
    day -= DaysBeforeMonths[i];
    month = i+1;
    break;
    }

}

GregDate = day +"." + month + "." + year;
return GregDate ;
```

With that being done, creating the inNdays() and ndaysago() methods was fairly easy. They both have 2 parameters, the first being the date and the second being the number N of days that you want to add/subtract. The methods would then call the calculateJD() method with the date as parameter, add/subtract N from the calculated Julian date and then convert the result of the addition/subtraction to gregorian using the jultogreg() method and return that.

```
@Override
public String ndaysago(int date, int n) {
    ndaysago = jultogreg(calculateJD(date) - n);
    return ndaysago;
}

@Override
public String inNdays(int date, int n) {
    inNdays = jultogreg(calculateJD(date) + n);
    return inNdays;
}
```

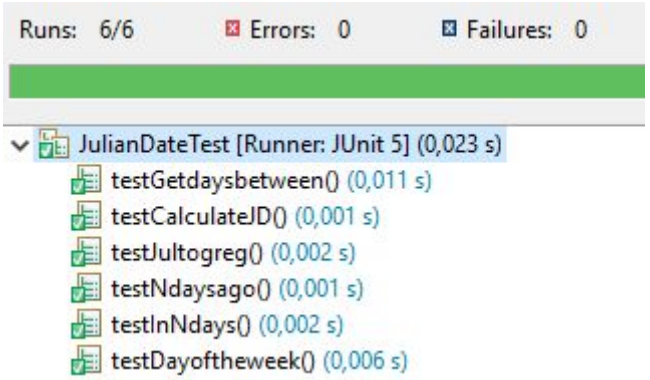
And finally we have the dayoftheweek() method using which we would be able to find the week-day of the inputted date. In the method we have a switch that has the the calculated julian date of the input modulo 7 + 1 as parameters, and has for each day of the week a case that sets the dayoftheweek String to the specific day.

```
@Override
public String dayoftheweek(int date) {

    switch((calculateJD(date)%7)+1)
    {
    case 1:
        dayoftheweek = "Monday";
        break;
    case 2:
        dayoftheweek = "Tuesday";
        break;
    case 3:
        dayoftheweek = "Wednesday";
        break;
    case 4:
        dayoftheweek = "Thursday";
        break;
    case 5:
        dayoftheweek = "Friday";
        break;
    case 6:
        dayoftheweek = "Saturday";
        break;
    case 7:
        dayoftheweek = "Sunday";
        break;
    }

    return dayoftheweek;
}
```

After having set up the methods, we now moved on to creating a testclass. We choose to create a JUnit class called JulianDateTest, since it is one of the best ways of testing code. It automatically created a test method for our JulianDate-methods, and we only had to add a few little things like our test cases. Once executed, the methods would compare the output from the respective JulianDate-methods to the expected output. After that a panel would show whether they failed or not.
We didn't use the test cases from the prelab, since they weren't suitable with our methods, as we had shifted the 0 from 4713BC to 01.01.1900.




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

import org.junit.jupiter.api.Test;

class JulianDateTest {


    @Test
    void testCalculateJD() {
        JulianDate test = new JulianDate();
        int output = test.calculateJD(19000101);
        assertEquals(0,output);
    }

    @Test
    void testGetdaysbetween() {

        JulianDate test = new JulianDate();
        int output = test.getdaysbetween(19000101, 20191108);
        assertEquals(43775,output);
    }

    @Test
    void testNdaysago() {
        JulianDate test = new JulianDate();
        String output = test.ndaysago(20191105, 30);
        assertEquals("6.10.2019",output);
    }

    @Test
    void testInNdays() {
        JulianDate test = new JulianDate();
        String output = test.inNdays(19000101, 30);
        assertEquals("31.1.1900",output);
    }

    @Test
    void testDayoftheweek() {
        JulianDate test = new JulianDate();
        String output = test.dayoftheweek(20191109);
        assertEquals("Saturday",output);
    }

    @Test
    void testJultogreg() {
        JulianDate test = new JulianDate();
        String output = test.jultogreg(43828);
        assertEquals("31.12.2019",output);
    }
}
```

2. Now make a little program that uses your Julian Date class. The program should ask for a birthday and figure out how many days old the person is and what weekday they were born on. If today is their birthday, then write out a special message. If you have lived a number of days that is divisible by 100, print a special message! Check your program using both of your birthdays. Which of you is the oldest? Is there a *Sunday's Child*?

To start things off we created a class called Birthday containing a main() method that throws IOException. In that class, we created a new object "jd" of type JulianDate and a new Date "date" which we formatted to yyyyMMdd. We then created a new BufferedReader "br" that has an InputStreamReader as parameter, which in turn has the System input as parameter.

```
public class Birthday {

    public static void main(String[] args) throws IOException {
        JulianDate jd = new JulianDate();

        Date date = new Date();
        SimpleDateFormat formatter = new SimpleDateFormat("yyyyMMdd");      //todays date
        int dateInt = Integer.parseInt( formatter.format(date));

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

After that we used the println() method to write out a message asking for the birthday. Following which, we read the input through the BufferedReader and parsed it to an Integer. Using the "jd" Object, we called the JulianDate-methods getdaysbetween() and dayoftheweek() with the input and "date" as parameters. The results from the methods then got outputted on the console.

```
System.out.println("What is you date of Birth?" + "\n" + "Please write it in the following format: yyyyMMdd");

int input = Integer.parseInt( br.readLine());

int InputDaysBetween = jd.getdaysbetween(input, dateInt);
String InputWeekday = jd.dayoftheweek(input);


System.out.println("You were born on a " + InputWeekday +", " + InputDaysBetween + " days ago!");
```

In case it was someone's birthday or the number of days they had lived until now was divisible by 100, we added 2 if-statements that would output a special message, if one of these situation was the case.

```
if (input%10000 == dateInt%10000) {     //if today is birthday
    System.out.println("Look at that, we've got a birthday kid! Here's a present for you: " + "\n" +" __OO__"
            + "\n" + "|  |  |" + "\n" + "|--|--|"+ "\n" + "|__|__|");
}

if (InputDaysBetween%100 ==0)           //if birthday divisible by 100
{
    System.out.println("That's exactly " + InputDaysBetween/100 + " times 100 days, today must be your lucky day!" );
}
```

Once executed it would look like this:

```
What is you date of Birth?
Please write it in the following format: yyyyMMdd
20191109
You were born on a Saturday, 0 days ago!
Look at that, we've got a birthday kid! Here's a present for you:
 __OO__
|  |  |
|--|--|
|__|__|
That's exactly 0 times 100 days, today must be your lucky day!
```

After trying out the program with our own birthdates, we got the following results:

```
<terminated> Birthday [Java Application] F:\Java\jdk-11.0
What is you date of Birth?
Please write it in the following format: yyyyMMdd
20000216
You were born on a Wednesday, 7206 days ago!
```

Bartholomäus

```
<terminated> Birthday [Java Application] F:\Java\jdk-11.(
What is you date of Birth?
Please write it in the following format: yyyyMMdd
19970524
You were born on a Saturday, 8204 days ago!
```

Akrem

As we can see Akrem is exactly 998 days older than Bartholomäus and none of us is a sunday's child.

3. A metric system is proposed to reform the calendar. It will have 10 regular days are a week, 10 weeks a month, 10 months a year. Extend your JulianDate class to a class MetricDate that has a method for converting from JulianDate to metric and from metric to JulianDate. How old are both of you on this metric system in years??

We first created a class MetricDate and then added the "extends MetricDate" to the JulianDate-class. After that, we created two methods, one that converts from Julian Date to Metric Date and one that does the opposite. We decided to keep the gregorian date format of not including the weeks themself into the date but adding them to days.

The first method gets the day, month and year in the new calendar using division and modulo. By dividing JD by 1000 we get the year, JD modulo 100 gives us the day number and the division of JD by 100 modulo 10 gets us the month.
We then convert all to string and we get the date in the Metric calendar.

```
public String calculateMD(int JD) {

    String calculateMD ;
    day = JD%100 ;
    month = (JD/100)%10 ;
    year = JD/1000;

    calculateMD = day + "." + month + "." + year ;
    return calculateMD;

}
```

In the second method called metricToJul, we choose to put 3 input parameters: MetricDay, MetricMonth and MetricYear. We then multiplied MetricYear by 1000 and MetricMonth by 100, and then added all 3 of them together to create the corresponding Julian Date.

```
public int metricToJul(int MetricDay, int MetricMonth, int MetricYear) {


    int JulianDate = MetricDay + MetricMonth*100 + MetricYear*1000;
    return JulianDate;
}
```

To get our ages in the metric date system, we called the getdaysbetween() method using our birthdays and today, we then used the result as parameters for the calculateMD() method and found out that Akrem is 8 years old and Bartholomäus is 7.

Reflections

Akrem:

The lab was a little bit challenging this time for many reasons.
First, we both missed the class of ADT, so we took time to get to the right way but i think that it was really an interesting lab. I liked working with my partner, which confirms the fact that team work has more advantages. I'm really looking for more challenging labs.

Bartholomäus:

As always the lab had its challenges, the first one being that I missed the classes on ADTs due to my bronchitis, which lead to some confusing hours while working on the prelab but I think that I have a good grasp on what they are or at least I hope that I do. The other ones were things like making small mistakes during coding, but nothing major. The only thing

I’m dissatisfied about this lab, was getting cursed by the phrase “Wednesday's child is full of woe”, it would be nice if that gets changed.

Code

```
public interface JD {

    public int calculateJD (int date);
    public int getdaysbetween (int date1, int date2);
    public String ndaysago (int date, int n);
    public String inNdays (int date, int n);
    public String dayoftheweek (int date);
    public String jultogreg (int date);

}
```

```
public class JulianDate extends MetricDate implements JD {

    private int JulianDate;
    private int daysbetween;
    private String ndaysago;
    private String inNdays;
    private int day;
    private int month;
    private int year;
    private String dayoftheweek;
    private String GregDate;
    private int [] DaysBeforeMonths = {0,31,59,90,120,151,181,212,243,273,304,334};
```

```
@Override
public int calculateJD(int date) {

    JulianDate = 0;


    day = date%100;
    month = (date-day)%10000/100;
    year = date/10000;

    year -= 1;
    JulianDate = day - 1;
    while (year >= 1900)
    {
        if ((year%4 ==0 && year%100 != 0) || (year%400 == 0))
        {
            JulianDate +=366;
        }
        else {
            JulianDate += 365;
        }
        year--;
    }

    JulianDate += DaysBeforeMonths[month-1];


        year = date/10000;
        if (((year%4 ==0 && year%100 != 0) || (year%400 == 0)) && month>2)
        {
            JulianDate +=1;
            }

    return JulianDate;
}

@Override
public int getdaysbetween(int date1, int date2) {
    if (calculateJD(date1) < calculateJD(date2))
    {
        daysbetween = calculateJD(date2) - calculateJD(date1);
    }
    else {
        daysbetween = calculateJD(date1) - calculateJD(date2);
    }
    return daysbetween;
}

@Override
public String ndaysago(int date, int n) {
    ndaysago = jultogreg(calculateJD(date) - n);
    return ndaysago;
}

@Override
public String inNdays(int date, int n) {
    inNdays = jultogreg(calculateJD(date) + n);
    return inNdays;
}
```

```
@Override
public String dayoftheweek(int date) {

    switch((calculateJD(date)%7)+1)
    {
    case 1:
        dayoftheweek = "Monday";
        break;
    case 2:
        dayoftheweek = "Tuesday";
        break;
    case 3:
        dayoftheweek = "Wednesday";
        break;
    case 4:
        dayoftheweek = "Thursday";
        break;
    case 5:
        dayoftheweek = "Friday";
        break;
    case 6:
        dayoftheweek = "Saturday";
        break;
    case 7:
        dayoftheweek = "Sunday";
        break;
    }

    return dayoftheweek;
}

@Override
public String jultogreg (int JulianDate)
{
    day = JulianDate+1;


    year = 1900;

    while (day > ((year%4 ==0 && year%100 != 0) || (year%400 == 0)?366:365))
    {
        day -= ((year%4 ==0 && year%100 != 0) || (year%400 == 0)?366:365);

        year++;
    }

    for (int i =11; i >-1; i--)
    {
        if (day >DaysBeforeMonths[i]) {
        day -= DaysBeforeMonths[i];
        month = i+1;
        break;
        }

    }

    GregDate = day +"." + month + "." + year;
    return GregDate ;

}
```

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

import org.junit.jupiter.api.Test;

class JulianDateTest {


    @Test
    void testCalculateJD() {
        JulianDate test = new JulianDate();
        int output = test.calculateJD(19000101);
        assertEquals(0,output);
    }

    @Test
    void testGetdaysbetween() {

        JulianDate test = new JulianDate();
        int output = test.getdaysbetween(19000101, 20191108);
        assertEquals(43775,output);
    }

    @Test
    void testNdaysago() {
        JulianDate test = new JulianDate();
        String output = test.ndaysago(20191105, 30);
        assertEquals("6.10.2019",output);
    }

    @Test
    void testInNdays() {
        JulianDate test = new JulianDate();
        String output = test.inNdays(19000101, 30);
        assertEquals("31.1.1900",output);
    }

    @Test
    void testDayoftheweek() {
        JulianDate test = new JulianDate();
        String output = test.dayoftheweek(20191109);
        assertEquals("Saturday",output);
    }

    @Test
    void testJultogreg() {
        JulianDate test = new JulianDate();
        String output = test.jultogreg(43828);
        assertEquals("31.12.2019",output);
    }
}
```

```
import java.io.*;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Birthday {

    public static void main(String[] args) throws IOException {
        JulianDate jd = new JulianDate();

        Date date = new Date();
        SimpleDateFormat formatter = new SimpleDateFormat("yyyyMMdd");      //todays date
        int dateInt = Integer.parseInt( formatter.format(date));

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        System.out.println("What is you date of Birth?" + "\n" + "Please write it in the following format: yyyyMMdd");

        int input = Integer.parseInt( br.readLine());

        int InputDaysBetween = jd.getdaysbetween(input, dateInt);
        String InputWeekday = jd.dayoftheweek(input);


        System.out.println("You were born on a " + InputWeekday +", " + InputDaysBetween + " days ago!");


        if (input%10000 == dateInt%10000) {     //if today is birthday
            System.out.println("Look at that, we've got a birthday kid! Here's a present for you: " + "\n" +" __OO__"
                    + "\n" + "|  |  |" + "\n" + "|--|--|"+ "\n" + "|__|__|");
        }

        if (InputDaysBetween%100 ==0)           //if birthday divisible by 100
        {
            System.out.println("That's exactly " + InputDaysBetween/100 + " times 100 days, today must be your lucky day!" );
        }
    }


}
```

```
public class MetricDate  {

    int day;
    int month;
    int year;
        public String calculateMD(int JD) {

            String calculateMD ;
            day = JD%100 ;
            month = (JD/100)%10 ;
            year = JD/1000;

            calculateMD = day + "." + month + "." + year ;
            return calculateMD;

        }

        public int metricToJul(int MetricDay, int MetricMonth, int MetricYear) {


            int JulianDate = MetricDay + MetricMonth*100 + MetricYear*1000;
            return JulianDate;
        }

}
```