Lab Report

EXERCISE 4: ABSTRACT DATA TYPES

Name		Titel des Kurses	Datum
Juri Wiechmann	571085	Prof. Dr. Weber-Wulff Info 2 Group 2	11.11.2019

Index

Introduction

This week's exercise was about Abtract Data Types and Interfaces. We were an odd number of students so I was free to be in a 3 man group or to be alone. I chose to work alone. I'm more efficient by working alone.

PRE-LAB

1. **How do Julien 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.**

```
package Übung4;

public interface JulianDate {

	public GregorDate JulianToGregor(int JD);
	public int daysBetween(GregorDate Day1, GregorDate Day2);
	public int GregorToJulian(GregorDate Day);
	public String get_Weekday(int JD);
}
```

Julian day is the continuous count of days since the beginning of the Julian Period and is used primarily by astronomers, and in software for easily calculating elapsed days between two events. 1.1.4712 BS is the most common day one(S1).

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.**

I decided now to have methods like tomorrow and yesterday because in JD it's the Day -1 for yesterday and +1 for tomorrow. It's not necessary to program a whole method for this. But as you can see in 1. I chose to have these 4.

Assignments

1. **For 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.**

In order of Object-oriented programming I decided to create the GregorDate Class which contains a lot of methods that will help us and some that I just wrote because I was lost in a rush and wanted everything that could fit into GregorDate to be there:

```
public class GregorDate {
    private int day;
    private int month;
    private int year;
    public Boolean AC;

    //month start with 0
    private static Calendar cal;

    public GregorDate(int year, int month, int day, Boolean AC) {
    public GregorDate() {
    public String toString() {
    public static int MonthsDays(int month, int year) {
    public GregorianCalendar toGregorianCalendar() {
        return new GregorianCalendar(this.year,this.month,this.day);
    }
    //First Day is day 1
    public int getDayOfTheYear() {
    public static GregorDate get_current_date() {
    //Gives us the Month in which the Day of the year is.
    public static int getMonthOfDayOfTheYear(int day, int year) {
    //Gives us the Day of any month by input the Day of the Year
    public static int DayOfYeartoDayofMonth(int day, int year) {
    public static boolean isLeapYear(int year) {
    //compares month and day of this Day and the param Day
    public boolean equalsMonth_Day(GregorDate Date2) {


    public int getDay() {
    public int getMonth() {
    public int getYear() {
    public void setDay(int day) {
    public void setMonth(int month) {
    public void setYear(int year) {

}
```

In AC we safe if the Date is After Christ or Before Christ. The rest should be clear by the names of the methods and if not there is a little. With these we could now go on to program the MyJulianDay class. The only 2 things we added which aren't in the interface were the field: `private static GregorDate FirstDay;` and the method:

```
public static GregorDate get_FirstDay() {
	return FirstDay;

}
```

In the contractor we gave it a value:

```
public MyJulianDate() {
	FirstDay = new GregorDate(4712,1,1, false);

}
```

Then I began to fill up our methods that they would do work. I started with the converting Methods. For this I used Wiki(S2):

```
@Override
public GregorDate JulianToGregor(int JD) {
	GregorDate outcome = new GregorDate();
	int JD0 = 1721426;
	int N400 = (JD-JD0)/146097;
	int R400 = (JD-JD0)%146097;
	int N100 = R400/36524;
	int R100 = R400%36524;
	int N4 = R100/1461;
	int R4 = R100%1461;
	int N1 = R4/365;
	int LT = R4%365;
	int LJ = 400*N400 + 100*N100 + 4*N4 + N1;
	outcome.setYear(LJ+1);
	outcome.setMonth((LT+1)/30 + 1);
	outcome.setDay(GregorDate.DayOfYeartoDayofMonth(LT + 1,
	outcome.getYear()));
	return outcome;
}
@Override
public int GregorToJulian(GregorDate Day) {
	int JD;
	int JD0 = 1721426;
	int LJ = Day.getYear()-1;
	int N400 = LJ/400;
	int R400 = LJ%400;
	int N100 = R400/100;
	int R100 = R400%100;
	int N4 = R100/4;
	int N1 = R100%4;
	int LT = Day.getDayOfTheYear()-1;
//-1 weil getDayOFTheYear bei 1 anfängt und nicht wie in der Formel bei 0
	JD = JD0 + N400*146097 + N100*36524 + N4*1461 + N1*365 + LT;

	return JD;
}
```

Then I wrote the get_Weekday Method which tells with the input of an JD the Weekday of this day.

```
public String get_Weekday(int JD) {

	int Weekday = (JD+1)%7;

	switch(Weekday) {
		case 0:
			return "Sunday";
		case 1:
			return "Monday";
		case 2:
			return "Tuesday";
		case 3:
			return "Wednesday";
		case 4:
			return "Thursday";
		case 5:
			return "Friday";
		case 6:
			return "Saturday";
	}
	return null;
}
```

The +1 is to synchronise our WeekDay-cycles with the real one.

Thanks to our converting Methods the daysBetween Method was really easy:

```
@Override
public int daysBetween(GregorDate Day1, GregorDate Day2) {

	int JDay1 = GregorToJulian(Day1);
	int JDay2 = GregorToJulian(Day2);
	return Math.abs(JDay2-JDay1);
}
```

Now in order to Test our Class I wrote in the main method every scenario that I had in mind that I could test. To Test every method with every Day by not overwriting the Test-Class I had the idea to ask for TestDates:

```
	GregorDate Test_Date1 = AskforDate();
	GregorDate Test_Date2 = AskforDate();
```

Ask for Dates then ask gently for input in our console. I also searched for a nice way to get input and found and used this(S3)

```
public static GregorDate AskforDate() {

	GregorDate outcome = new GregorDate();
	System.out.println("DayBetween TestDate Year(YYYY):");
	outcome.setYear(in.nextInt());
	System.out.println("DayBetween TestDate Month(MM):");
	outcome.setMonth(in.nextInt());
	System.out.println("DayBetween TestDate Month(DD):");
	outcome.setDay(in.nextInt());

	return outcome;
}
```

```
DayBetween TestDate Year(YYYY):
1998
DayBetween TestDate Month(MM):
02
DayBetween TestDate Month(DD):
19
DayBetween TestDate Year(YYYY):
2019
DayBetween TestDate Month(MM):
11
DayBetween TestDate Month(DD):
11
```

Now I went to test every Method that I wrote so far:

```
System.out.println("Between " + Test_Date1 + " and " + Test_Date2 + " are " +
MJD.daysBetween(Test_Date1, Test_Date2) + " days." );

	//Testing Julien to Gregor and Gregor to Julien:
	System.out.println(GregorDate.get_current_date());

System.out.println(MJD.JulianToGregor(MJD.GregorToJulian(GregorDate.get_curr
ent_date())));
	//toString is automaticly used!!!

	//Testing Weekdays:
	//Today:

System.out.println(MJD.get_Weekday(MJD.GregorToJulian(GregorDate.get_current
_date())));
	//next few Days with Date that u can see its right
	for(from CurrentDate on 14 days to the future){
		System.out.println(MJD.get_Weekday(i));
		System.out.println(MJD.JulianToGregor(i));
	}
	//for a specific day to be sure:


System.out.println(MJD.get_Weekday(MJD.GregorToJulian(AskforDate())));

	//First Julien Day:
	System.out.println(MJD.get_FirstDay());
```

```
Between 1998.02.19 and 2019.11.11 are 7935 days.
2019.11.11
2019.11.11
Monday
```

```
DayBetween TestDate Year(YYYY):
2022
DayBetween TestDate Month(MM):
12
DayBetween TestDate Month(DD):
24
Saturday
```

First I tested the days.Between Method. Then I printed the current Date directly and then I converted it 2 times to be sure that's the converting is right. In this moment I was surprised because I don't have to use the toString() Method. Its automatically used the toString()-Method in print methods. Then I started to Test the Weekday Method. First the current day and then in the loop all of the next 14 Days. But I wanted to be very sure that its even right in years and asked again for a Test-Date:

```
Monday
2019.11.11
Tuesday
2019.11.12
Wednesday
2019.11.13
Thursday
2019.11.14
Friday
2019.11.15
Saturday
```

Last but least the First-Day method: 4712.01.01 ←output

2. Afdgdgw 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?

Here I just created 2 field with by GregorDate-class and MyJulianDate-class. The GregorDate is to safe our Birthday which we ask for in the Constructor:

```
private static GregorDate Birthday = new GregorDate(1, 1, 1, true);

private static MyJulianDate MJD = new MyJulianDate();

public Birthday(GregorDate Birthday) throws IOException {
	this.Birthday = Birthday;
	main(null); //its not pretty i know....
}
```

The first thing I leaned here were that we don't call the main method by creating an Object, for this example later in our Test-class. It drove me crazy why my Birthday-class don't work till after a lot of time I found it out. Because I was a bit mad I forced him to call the main-method which do will do all the things that the Assignment is asking for:

```
public static void main(String[] args) throws IOException {

	if(Birthday.equalsMonth_Day(GregorDate.get_current_date()))
		System.out.println("HAPPY BIRTHDAY");
	if( MJD.daysBetween(Birthday,GregorDate.get_current_date())%100 == 0)
		System.out.println("You lived a number of days that is
		divisible by 100.");

	System.out.println("You are " +
	MJD.daysBetween(Birthday,GregorDate.get_current_date()) + " days
	old");
	System.out.println("You were born on a " +
	MJD.get_Weekday(MJD.GregorToJulian(Birthday)));

}
```

Now I tested it in our Test-class by asking again for a Test-Date:

```
	//Test Birthday
	Birthday MyBirthday = new Birthday(AskforDate());
```

Now I simulated like I would have my Birthday today and if it would be dividable by 100:

```
DayBetween TestDate Year(YYYY):
1998                           1998
DayBetween TestDate Month(MM): DayBetween TestDate Month(MM):
11                             10
DayBetween TestDate Month(DD): DayBetween TestDate Month(DD):
11                             12
HAPPY BIRTHDAY                 You lived a number of days that is divisible by 100.
You are 7670 days old          You are 7700 days old
You were born on a Wednesday   You were born on a Monday
```

Yeah it worked nice.

3. **Ametric 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??**

First I created 4 fields. In mind that we can now count weeks too I thougt we also should count them.

```
private int day;
private int week;
private int month;
private int year;
```

Then I started to think about converting from Julian to Metric and back. Its not that much of a deal but you need to keep track on one thing. In Juliandates there are not 10 Day or Weeky to count. All time they reached 10 they jump to a 0 and ad one to the next digit. In our Metric-system we would like to count to 10 Days and then count at 11 Day 1 to the weeks. Then so start with 1 again. Normal peaople would just add 1 if they need to show the Date. But I thougt way to complicated and also wanted not only to show them but also so safe and deal with them only with digits from 1-10 and not 0-9. In order to do so I developed this Way:

```
public MetricDate JulianToMetric(int JD) {
	day = ((JD%10==0)? 10 : JD%10);
	JD -= day;
	week = ((JD%100/10==0)? 10 : JD%100/10);
	JD -= week*10;
	month = ((JD%1000/100==0)? 10 : JD%1000/100);
	JD -= month*100;
	year = JD/1000;
	return this;
}
```

And to convert back we just need to add every field in the right digit field:

```
public int MetricToJulian(MetricDate MD) {

	return day+week*10+month*100+year*1000;
}
```

Now I wanted to create some constructors for the MetricDate-class where u can put in different types:

```
//if u know the param
public MetricDate(int year, int month, int week, int day) {
	super();
	setDay(day);
	setWeek(week);
	setMonth(month);
	setYear(year);
}
//with JulienDate
public MetricDate(int JD) {
	super();

	MetricDate MD = JulianToMetric(JD);
	setDay(MD.getDay());
	setWeek(MD.getWeek());
	setMonth(MD.getMonth());
	setYear(MD.getYear());
}
//With the GregorDates
public MetricDate(GregorDate G_Date) {
	super();
	MetricDate MD = new MetricDate(GregorToJulian(G_Date));
	setDay(MD.getDay());
	setWeek(MD.getWeek());
	setMonth(MD.getMonth());
	setYear(MD.getYear());
}
```

I took a look at the last few assignments and I m not sure but I guess I accidentally solved 9 and 10 too. I wrote just for comfort toString-Methods to have a better testing-output and for 9 maybe my GregorDate is this ISO standard.

Reflections

JURI

In this weeks`s lab I worked the first time alone. It have positive Sites and negative one. You are for your own and more productive. You don't have to arrange with other or waiting for their response. On the other hand its simply more work and none take a look at what you did before the postlab get graded. On the programing site I fall instantly in love with this ()bool)? True : false) structure that I overheard in the lecture. I used them here and in GDM often, maybe to even to often. I also created a UML diagram to present a better overview but the website crashed as I wanted to safe it and is very user unfriendly (S4).

S1: https://en.wikipedia.org/wiki/Julian_day

S2: https://de.wikipedia.org/wiki/Umrechnung_zwischen_julianischem_Datum_und_gregorianischem_Kalender

S3: https://data-flair.training/blogs/read-java-console-input/

S4: http://uml.bozeman.de/uml.php5

Appendix:

Java Code:

```
package Übung4;

public interface JulianDate {

	public GregorDate JulianToGregor(int JD);
	public int daysBetween(GregorDate Day1, GregorDate Day2);
	public int GregorToJulian(GregorDate Day);
	public String get_Weekday(int JD);
}
```

```java
package Übung4;


public class MyJulianDate implements JulianDate {

	private static GregorDate FirstDay;



	public MyJulianDate() {
		FirstDay = new GregorDate(4712,1,1, false);
	}
	@Override
	public int daysBetween(GregorDate Day1, GregorDate Day2) {

		int JDay1 = GregorToJulian(Day1);
		int JDay2 = GregorToJulian(Day2);
		return Math.abs(JDay2-JDay1);
	}
	@Override
	public GregorDate JulianToGregor(int JD) {

		GregorDate outcome = new GregorDate();

		int JD0 = 1721426;
		int N400 = (JD-JD0)/146097;
		int R400 = (JD-JD0)%146097;
		int N100 = R400/36524;
		int R100 = R400%36524;
		int N4 = R100/1461;
		int R4 = R100%1461;
		int N1 = R4/365;
		int LT = R4%365;

		int LJ = 400*N400 + 100*N100 + 4*N4 + N1;

		outcome.setYear(LJ+1);
		outcome.setMonth((LT+1)/30 + 1);
		outcome.setDay(GregorDate.DayOfYeartoDayofMonth(LT + 1,
outcome.getYear()));

		return outcome;
	}
	//https://de.wikipedia.org/wiki/Umrechnung_zwischen_julianischem_Dat
um_und_gregorianischem_Kalender
	@Override
	public int GregorToJulian(GregorDate Day) {
		int JD;
		int JD0 = 1721426;
		int LJ = Day.getYear()-1;
		int N400 = LJ/400;
		int R400 = LJ%400;
		int N100 = R400/100;
		int R100 = R400%100;
```

```
		int N4 = R100/4;
		int N1 = R100%4;
		int LT = Day.getDayOfTheYear()-1;

		//-1 weil getDayOFTheYear bei 1 anfängt und nicht wie in der
Formel bei 0
		JD = JD0 + N400*146097 + N100*36524 + N4*1461 + N1*365 + LT;

		return JD;
	}
	@Override
	public String get_Weekday(int JD) {

		int Weekday = (JD+1)%7;

		switch(Weekday) {
			case 0:
				return "Sunday";
			case 1:
				return "Monday";
			case 2:
				return "Tuesday";
			case 3:
				return "Wednesday";
			case 4:
				return "Thursday";
			case 5:
				return "Friday";
			case 6:
				return "Saturday";
		}
		return null;
	}
	public static GregorDate get_FirstDay() {
		return FirstDay;
	}

}
```

```
package Übung4;

import java.util.Calendar;
import java.util.GregorianCalendar;

public class GregorDate {
	private int day;
	private int month;
	private int year;
	public Boolean AC;

	//month start with 0
	private static Calendar cal;

	public GregorDate(int year, int month, int day, Boolean AC) {
		super();

		this.cal = Calendar.getInstance();
		this.AC = AC;

		setYear(year);;
		setMonth(month);
		setDay(day);
	}
	public GregorDate() {
```

```
	}
	@Override
	public String toString() {
		String FirstLetterDay = ((this.day<10) ? "0" : "");
		String FirstLetterMonth = ((this.month<10) ? "0" : "");
		return Integer.toString(year) + "." + FirstLetterMonth +
Integer.toString(month) + "." + FirstLetterDay + Integer.toString(day);
	}
	public static int MonthsDays(int month, int year) {

		if(!(0 < month && month < 13))
			throw new IllegalArgumentException("months out of range");

		switch(month) {
		case 4:
			return 30;
		case 6:
			return 30;
		case 9:
			return 30;
		case 11:
			return 30;

		case 2:
			return ((isLeapYear(year)) ? 29 : 28);
		}
	   return 31;
	}
```

```
	public GregorianCalendar toGregorianCalendar() {
		return new GregorianCalendar(this.year,this.month,this.day);
	}
	//First Day is day 1
	public int getDayOfTheYear() {

		int dayOfTheYear = 0;

		for(int i = 1; i<month;i++) {
			dayOfTheYear += MonthsDays(i, this.year);
		}
		return dayOfTheYear + day;
	}
	public static GregorDate get_current_date() {

		//Here we want month + 1 cuz we our Date start with month 1
		GregorDate outcome = new
GregorDate(cal.get(Calendar.YEAR),cal.get(Calendar.MONTH) +
1,cal.get(Calendar.DAY_OF_MONTH),true);
		return outcome;
	}
	//Gives us the Month in which the Day of the year is.
	public static int getMonthOfDayOfTheYear(int day, int year) {
		int month = 1;
		for(; MonthsDays(month, year) < day; month++) {
			day -= MonthsDays(month, year);
		}
```

```
		return month;
	}
	//Gives us the Day of any month by input the Day of the Year
	public static int DayOfYeartoDayofMonth(int day, int year) {
		int month = 1;
		for(; MonthsDays(month, year) < day; month++) {
			day -= MonthsDays(month, year);
		}

		return day;
	}
	public static boolean isLeapYear(int year) {
		return (((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) ? true :
false);
	}
	//compares month and day of this Day and the param Day
	public boolean equalsMonth_Day(GregorDate Date2) {

		if(this.month == Date2.getMonth() && this.day == Date2.getDay())
			return true;
		else
			return false;

	}
```

```
	public int getDay() {
		return day;
	}
	public int getMonth() {
		return month;
	}
	public int getYear() {
		return year;
	}
	public void setDay(int day) {
		if(0 < day && day <= MonthsDays(month, year))
			this.day = day;
		else
			throw new IllegalArgumentException("days out of range");

	}
	public void setMonth(int month) {
		if(0 < month && month < 13)
			this.month = month;
		else
			throw new IllegalArgumentException("months out of range");
	}
	public void setYear(int year) {
		this.year = year;
	}

}
```

```
package Übung4;


public class MetricDate extends MyJulianDate{


	private int day;
	private int week;
	private int month;
	private int year;


	//if u know the param
	public MetricDate(int year, int month, int week, int day) {
		super();
		setDay(day);
		setWeek(week);
		setMonth(month);
		setYear(year);
	}
	//with JulienDate
	public MetricDate(int JD) {
		super();

		MetricDate MD = JulianToMetric(JD);
		setDay(MD.getDay());
```

```
        setWeek(MD.getWeek());
        setMonth(MD.getMonth());
        setYear(MD.getYear());
}
//With the GregorDates
public MetricDate(GregorDate G_Date) {
        super();
        MetricDate MD = new MetricDate(GregorToJulian(G_Date));
        setDay(MD.getDay());
        setWeek(MD.getWeek());
        setMonth(MD.getMonth());
        setYear(MD.getYear());
}
public MetricDate JulianToMetric(int JD) {

        day = ((JD%10==0)? 10 : JD%10);
        JD -= day;

        week = ((JD%100/10==0)? 10 : JD%100/10);
        JD -= week*10;

        month = ((JD%1000/100==0)? 10 : JD%1000/100);
        JD -= month*100;

        year = JD/1000;

        return this;
}
```

```
public int MetricToJulian(MetricDate MD) {

        return day+week*10+month*100+year*1000;
}

public String toString() {
        String FirstLetterDay = ((this.day<10) ? "0" : "");
        String FirstLetterWeek = ((this.week<10) ? "0" : "");
        String FirstLetterMonth = ((this.month<10) ? "0" : "");
        return Integer.toString(year) + "." +
                        FirstLetterMonth + Integer.toString(month) + "." +
                        FirstLetterWeek + Integer.toString(week) + "." +
                        FirstLetterDay + Integer.toString(day);
}


public int getDay() {
        return day;
}
public int getWeek() {
        return week;
}
public int getMonth() {
        return month;
}
public int getYear() {
        return year;
}
```

```
	public void setDay(int day) {
		if( 0 < day && day <= 10)
			this.day = day;
		else
			throw new IllegalArgumentException("days out of range");
	}
	public void setMonth(int month) {
		if( 0 < month && month <= 10)
			this.month = month;
		else
			throw new IllegalArgumentException("months out of range");
	}
	public void setWeek(int week) {
		if( 0 < week && week <= 10)
			this.week = week;
		else
			throw new IllegalArgumentException("weeks out of range");
	}
	public void setYear(int year) {
		this.year = year;
	}



}
```

```
package Übung4;

import java.io.IOException;
import java.util.Scanner;

public class Test {

	//https://data-flair.training/blogs/read-java-console-input/
	private static Scanner in = new Scanner(System.in);

	public static void main(String[] args) throws IOException {

		MyJulianDate MJD = new MyJulianDate();



		//Test JulianDAte

		/*
		//Days Between:

		GregorDate Test_Date1 = AskforDate();
		GregorDate Test_Date2 = AskforDate();

		System.out.println("Between " + Test_Date1 + " and " + Test_Date2 + "
are " + MJD.daysBetween(Test_Date1, Test_Date2) + " days." );
```

```
		//Testing Julien to Gregor and Gregor to Julien:
		System.out.println(GregorDate.get_current_date());

	System.out.println(MJD.JulianToGregor(MJD.GregorToJulian(GregorDate.get_
current_date())));
		//toString is automaticly used!!!

		//Testing Weekdays:
		//Today:

	System.out.println(MJD.get_Weekday(MJD.GregorToJulian(GregorDate.get_cu
rrent_date())));
		//next few Days with Date that u can see its right
		for(int i = MJD.GregorToJulian(GregorDate.get_current_date()); i< 15 +
MJD.GregorToJulian(GregorDate.get_current_date()); i++){
			System.out.println(MJD.get_Weekday(i));
			System.out.println(MJD.JulianToGregor(i));
		}
		//for a specific day to be sure:

	System.out.println(MJD.get_Weekday(MJD.GregorToJulian(AskforDate())));

		//First Julien Day:
		System.out.println(MJD.get_FirstDay());

		//Test Birthday
		Birthday MyBirthday = new Birthday(AskforDate());
```

```
		*/
		//Test MeticDate with Birthday:
		//3 Way to Create MD:
		//Gregor:
		MetricDate MD = new MetricDate(new GregorDate(1998,02,19,true));
		System.out.println(MD);
		//direct:
		MD = new MetricDate(2450,8,6,4);
		System.out.println(MD);
		//with Julian:
		MD = new MetricDate(MD.GregorToJulian(new
GregorDate(1998,02,19,true)));
		System.out.println(MD);

		System.out.println(123450);
		System.out.println(MD.JulianToMetric(123450));
		System.out.println(MD.MetricToJulian(MD.JulianToMetric(123450)));
	}
	public static GregorDate AskforDate() {
		GregorDate outcome = new GregorDate();
		System.out.println("DayBetween TestDate Year(YYYY):");
		outcome.setYear(in.nextInt());
		System.out.println("DayBetween TestDate Month(MM):");
		outcome.setMonth(in.nextInt());
		System.out.println("DayBetween TestDate Month(DD):");
		outcome.setDay(in.nextInt());

		return outcome;
```

```
	}
}
package Übung4;

import java.io.IOException;
import java.util.Scanner;



public class Birthday {


	private static GregorDate Birthday = new GregorDate(1, 1, 1, true);



	private static MyJulianDate MJD = new MyJulianDate();

	public Birthday(GregorDate Birthday) throws IOException {
		this.Birthday = Birthday;
		main(null); //its not pretty i know....
	}

	public static void main(String[] args) throws IOException {


		if(Birthday.equalsMonth_Day(GregorDate.get_current_date()))
			System.out.println("HAPPY BIRTHDAY");
		if( MJD.daysBetween(Birthday,GregorDate.get_current_date()) % 100
== 0)
```

```
                    System.out.println("You lived a number of days that is divisible
by 100.");


                System.out.println("You are " +
MJD.daysBetween(Birthday,GregorDate.get_current_date()) + " days old");

                System.out.println("You were born on a " +
MJD.get_Weekday(MJD.GregorToJulian(Birthday)));


        }

}
```