# Workday calendar

This code kata assignment assumes development in the Java language, but you may adapt it to C# or the language of your choice.

We need a class which calculates working days:

```
WorkdayCalendar workdayCalendar = new WorkdayCalendar();
```

The class will calculate distance measured in working days from a given date (a `java.util.Date`).
A working day is defined as a day from Monday to Friday which is not a holiday. WorkdayCalendar has 2 methods which tell which days shall be regarded as holidays:

```
workdayCalendar.setHoliday(Calendar date)
```

says that the given date is a holiday and shall not count as a working day.

```
workdayCalendar.setRecurringHoliday(Calendar date)
```

says the given date is to be regarded as a holiday on the same date every year (in other words disregard the year component).

```
workdayCalendar.setWorkdayStartAndStop(Calendar start, Calendar stop)
```

sets the start and stop times for the working day, e.g. 8-16 (all examples are provided in Java, your mileage may vary depending upon your language of choice):

```
workdayCalendar.setWorkdayStartAndStop(
    new GregorianCalendar(2004, Calendar.JANUARY, 1, 8, 0),
    new GregorianCalendar(2004, Calendar.JANUARY, 1, 16, 0))
```

Here we must disregard the given date.

The heart of the solution is the following method:

```
public Date getWorkdayIncrement(Date startDate, float incrementInWorkdays)
```

The method must always return a time which is between the two times defined in the method `setWorkdayStartAndStop`, even though the `startDate` need not follow this rule. According to this rule then 15:07 + 0.25 working days will be 9:07, and 4:00 plus 0.5 working days will be 12:00.

The class can be tested with e,g. code like the following:

```
WorkdayCalendar workdayCalendar = new WorkdayCalendar();
workdayCalendar.setWorkdayStartAndStop(
    new GregorianCalendar(2004, Calendar.JANUARY, 1, 8, 0),
    new GregorianCalendar(2004, Calendar.JANUARY, 1, 16, 0));

workdayCalendar.setRecurringHoliday( new GregorianCalendar(2004, Calendar.MAY, 17,
0, 0));
workdayCalendar.setHoliday( new GregorianCalendar(2004, Calendar.MAY, 27, 0, 0));

SimpleDateFormat f = new SimpleDateFormat("dd-MM-yyyy HH:mm");
Date start = new GregorianCalendar(2004, Calendar.MAY, 24, 18,5).getTime();
float increment = -5.5f;
System.out.println(f.format(start) +" with the addition of " +
  increment + " working days is " +
  f.format(workdayCalendar.getWorkdayIncrement(start, increment)));
```

... which should give the following result:

```
24-05-2004 18:05 with the addition of -5.5 working days is 14-05-2004 12:00
```

Some other correct results:

```
24-05-2004 19:03 with the addition of 44.723656 working days is 27-07-2004 13:47
24-05-2004 18:03 with the addition of -6.7470217 working days is 13-05-2004 10:02
24-05-2004 08:03 with the addition of 12.782709 working days is 10-06-2004 14:18
24-05-2004 07:03 with addition of 8.276628 working days is 04-06-2004 10:12
```

The solution should be a single class file, possibly with inner classes if your solution uses them. The best solutions are easy to read, relatively short (under 250 lines of code), and should of course calculate the correct answer for a wide range of cases.

Please submit complete, runnable project with all required files as an invitation to clone your private git-repository. Email with Zip-file attached is considered bad form these days.

Some additional instructions depending upon your language of choice:

- C++ - you may only use the standard libraries supplied togehter with the compiler. Only exception is unit-testing libaries.
- Java - please use Maven

Good luck! *****

## Evaluation criteria

- Level of understanding the assignment in terms of the explicit and implicit information given. Hint: decision matrix.
- Communication of the implemented solution
- Coding competency, i.e. TDD, DRY, SOLID, modularization of the domain, etc.
- The layout of source files, build files and other supporting files.