Assignment 1: Basic Squeak

Introduction
1. In this assignment you will write an application for building an academic weekly time-table. In order to do this, you must write two classes as described below. These classes must belong to a new category called “OOP1” and must inherit from the class Object.
2. Objective of this assignment: A basic acquaintance with Squeak. This assignment is meant to greatly help you to cope with the following assignments.
3. In order to avoid mistakes please read the FAQ section in the course website which will be updated on an ongoing basis.
4. For this assignment, please use Squeak version 4.5, which can be downloaded from the official site.
5. The TA in charge of this assignment is Eliran Weiss. Questions should be sent to the mail eliranw@cs.technion.ac.il with the subject “236703 HW1”.
6. The assignment is due on 19.4.15 at 23:55.

Class Details
1. The class Course defines an academic course. Courses will be considered identical if they have the same course number. Every course is given in exactly one day and at a certain time, and can last either an hour or two. For this exercise, we'll use a rather simple date & time format, but it's worth noticing that Squeak comes with many classes for date & time representation.

Instance Variables:
- **id** – Holds a string that represents the course number. Initialized to ‘0’ (a string consisting of the character zero).
- **name** – Holds the name of the course. Initialized to the empty string.
- **day** – Holds the index of the day of the week in which the course is given. Possible values for this field are integers between 1 and 5. Initialized to 1 (Sunday).
- **time** – Holds the starting time of the course lecture. In order to simplify things, we will assume that courses start on the hour, and so the hour will be represented by an integer between 8 and 20. Initialized to 8.
- **isTwoHoursLong** – Holds a flag indicating if the course is two hours long. If this value is false then the course length is one hour. (For the purposes of this assignment a course can only be either one or two hours long). Initialized to false.

Instance Methods
- **id** – Returns the value of the field id.
- **name** – Returns the value of the field name.
• **day** – Returns the value of the field *day*.

• **time** – Returns the value of the field *time*.

• **isTwoHoursLong** – Returns the value of the field *isTwoHoursLong*.

• **id: aString** – Sets the course number. If the value which the method receives is not a string or is a string that doesn’t represent a number then the method must throw an exception with the string ‘invalid input value’. The exception should be thrown using the method *error:* (the method is defined in *Object*).

• **name: aString** – Sets the name of the course. If the value which the method receives is not a string then the method must throw an exception with the string ‘invalid input value’. Use the method *error:* to throw the exception.

• **day: anInteger** – Sets the day of the week in which the course is given. If the value which the method receives is not an integer within the legal range the method must throw an exception with the string ‘invalid input value’. Use the method *error:* to throw the exception.

• **time: anInteger** – Sets the time of day of the course. If the value which the method receives is not an integer within the legal range the method must throw an exception with the string ‘invalid input value’. Use the method *error:* to throw the exception.

• **isTwoHoursLong: aBoolean** – Sets the flag indicating whether the class is one or two hours long. If the value which the method receives is not a Boolean the method must throw an exception with the string ‘invalid input value’. Use the method *error:* to throw the exception.

2. The class *WeeklySchedule* defines the weekly schedule.

**Instance Variables**

• **courses** – Holds a collection of objects of type *Course*. You must choose the collection that best fits the requirements.

**Instance Methods**

• **addCourse: aCourse** – The method adds the course to the system if possible. A course may **not** be added if its lecture time overlaps the lecture time of a course already in the system, or if a course with the same course number already exists in the system. If the course cannot be added for one of these reasons the method must throw an exception with the string ‘cannot add course’. Use the method *error:* to throw the exception.

*You should prevent changes in added courses outside the WeeklySchedule class, since this is a common cause for bugs and unexpected behavior.*

• **removeCourse: aString** – The method will remove the course with the number received as a parameter from the system. If such a course doesn’t exist the method will do nothing.

• **totalHours** – The method returns the total number of hours of courses in the schedule.

• **compareTo: aSchedule** – The method compares between the current WeeklySchedule and the WeeklySchedule received as a parameter. It returns *true* if the schedules
contain the exact same courses, and \textit{false} otherwise. Remember that courses are considered the same if and only if the value of their \texttt{id} field is equal, and therefore it's certainly possible that the same course will be given in different times (but not twice in the same schedule).

- **filterCourses: aBlock** -
  - The method receives a block that receives a single parameter of type Course and returns Boolean value.
  - The method must return a block that is an iterator over all courses in the system for which aBlock returns \textit{true}
    - The courses returned by the iterator should be sorted according to the day and hour of the course. For example, each time the \textit{value} method is sent to the iterator block, the return value will be the next course in the sub-collection that was received by applying aBlock to the courses in the schedule (aBlock works as a filter).
    - If the iterator has already traversed all of the filtered courses the iterator block will return \textit{nil}.
    - Important: The iterator works on a snapshot of the system at the moment \textit{filterCourses:} was called. Changes to the schedule after calling \textit{filterCourse:} will not be shown by the iterator. (See examples below).

- **coursesOnDate: aDate** - The method receives an object of the type \texttt{Date} (A class belonging to the Squeak library). The method calculates the corresponding weekday and returns a block that is an iterator over all of the courses in the system that occur on that day of the week. The order of courses and limitations on the iterator are the same as described in the method \textit{filterCourses:}. You may assume that the date received is legal and corresponds to a day of the week between Sunday and Thursday. It is recommended that you use the method \textit{filterCourses:} to implement this method.

### Run Examples:
For your convenience some examples are presented here. These examples are only to assist you in writing the exercise, and do not cover all required functionality.

```
<table>
<thead>
<tr>
<th>c1 c2 c3 c4 c5 ws ws2 b1 iter1 iter2</th>
</tr>
</thead>
</table>
c1 := Course new id: '236703'; name: 'OOP'; day:2; time:14; isTwoHoursLong: true.
c2 := Course new id: '234114'; name: 'intro to CS'; day:2; time:10; isTwoHoursLong: true.
c3 := Course new id: '1337'; name: 'intro to RE'; day:5; time:10; isTwoHoursLong: false.
c4 := Course new id: '01234'; name: 'Physics 2P'; day:1; time:17; isTwoHoursLong: true.
```
ws := WeeklySchedule new.
ws addCourse: c1.
ws addCourse: c2.
ws addCourse: c3.
ws addCourse: c4.
Transcript show: ws totalHours; cr.
b1 := [: c | c isTwoHoursLong].
iter1 := ws filterCourses: b1.
ws removeCourse: '236703'.
Iter2 := ws filterCourses: b1.

Transcript show: iter1 value id; cr.
Transcript show: iter1 value id; cr.
Transcript show: iter1 value id; cr.
Transcript show: iter1 value; cr.

Transcript show: iter2 value id; cr.
Transcript show: iter2 value id; cr.
Transcript show: iter2 value; cr.

c5 := Course new id: '234114'; name: 'fake CS'; day: 4;
time: 19; isTwoHoursLong: false.
ws2 := WeeklySchedule new.
ws2 addCourse: c5.
ws2 addCourse: c3.
ws2 addCourse: c4.
Transcript show: (ws2 compareTo: ws); cr.
Notes

1. Squeak’s default equals operator (=) compares objects by reference. We will learn more about this operator later on in the course; however, if you wish to use collections such as Dictionary or Set, which use comparisons of objects in their implementation, you will have to implement the operator = and also the method hash for the object which will be compared (if = and hash are not yet implemented for this object type). Implementation examples of these methods for the point2D class, that contains x and y fields:

```plaintext
= anObject
  ^(anObject class = Point2D) and: [(x = anObject getX) and: [y = anObject getY]]
```

```plaintext
hash
  ^(x + y) hash
```

Since various classes (particularly Sets and Dictionaries) rely on the property that equal objects have equal hashes, you should override hash whenever you override =; it must be true that (a = b) implies (a hash = b hash). The contrapositive and the converse will not generally hold.


2. In methods where it’s not specifically mentioned, you do not need to check for parameters correctness.

3. In methods where you are required to return an error with a string, please make sure you are returning the exact string with the correct capitalization and whitespacing.

4. Execution speed is not a central issue in the homework assignments in this course. In any case of uncertainty between simplicity and performance, choose the simpler implementation. Objections such as: “I preferred a faster implementation so the code is less clear” will not be accepted.

5. Care should be taken that names of classes, methods and fields are exactly as defined in this document. You may add as many helper methods and fields as you like.

6. Your code should be clear, readable, and commented to a reasonable degree. Likewise, try to avoid code duplication – use methods you’ve already written as much as possible.

7. It is recommended that you read the first two recitations in Squeak and the SqueakLab before starting work.

8. Comment every non-trivial part of the code.

9. For your convenience, some common errors:
   - Not using the keyword self when an object sends a message to itself (This is different from C++ where this may be omitted).
   - Omitting a period (.) after a statement in the body of a method/in a block.
   - When using Boolean expressions (such as the messages or:, and:) The right-hand statement must be a block that holds a boolean expression. Example: x>y and: [x>z].
• It is not possible to assign new values into a method parameter. For example, if \( x \) is a parameter to a method then the assignment \( x := x \times 2 \) will not change the value of \( x \). The solution: assign into a local variable.

• To use the collection LinkedList you must use an object that inherits from the class Link. However, as mentioned, all implemented classes should inherit from Object. Therefore you should use OrderedCollection instead, which does not require inheritance from another class. (This is more correct from an OO perspective if the element held in the list does not always exist only as part of the list).

• If your program gets stuck, use alt+.’ to create a user interrupt that will stop the program’s run and allow you to see where it got stuck.

• Blocks in Squeak are closures, and have special properties. For specific details, read the first two recitations carefully.

• Squeak works with references! Remember that when you enter an object into a collection, you are actually entering a reference to it.

Submission Details

• Requests for postponement, for any reason, must be sent to the TA in charge of the course (Nurit). Note that the course has a late-submission policy, meaning you can submit your assignment late without a postponement approval. Details can be found in the course site under “General Info”.

• The assignment must be submitted electronically. (Save the submission confirmation).

• Your submission should consist of a zip file with the name format: OOP1_<ID1>_._<ID2>.zip containing:
  o A file called readme.txt with the name, id and email of each of the submittors.
  o The code file: OOP1.st. The file must contain only the implementations of the classes required in the assignment and methods needed for the implementation. Do not hand in extra code, such as tests.

• Points will be deducted for not conforming to the submission requirements (rar instead of zip, extra files, a readme file with the wrong name, etc.

Good Luck!