# Advanced JavaScript Practical Exam: School System

The goal of this sample exam is to practice the creation of a JavaScript structure, using the best practices of **quality JavaScript**.

### The School System

**The School system** implements the operating system of a typical school. The School System should have Principals, Teachers, Students and Grades.

### The Source Code

You are given the following components:

* **index.html** - The HTML file
* **libs folder** - The libraries of application;
* **helpers folder** – Additional helper files to implement (if needed);
* **models folder** - The models you need to implement;
* **services folder** – The services that operate over the models
* **generator.js** – Generates test data;
* **app.js** – Fills the tables with data;

Your task is to implement only the missing **models**. If you implement them correctly, your application will run without errors. You **do not need** to edit the **generator.js** and **app.js** files.

## Implement function constructors or objects for all types

You can use Pseudo-classical or Prototypal OOP. All **modules** should have **visible (not hidden) getter** and **setters** (functions that set a certain value to a property with validation or throws an error) for all hidden (with underscore) properties.

1. Human – Human is an **abstract** (no initialization should be possible) module and should have:

Constructor:

* + name parameter – The constructor should have only 1 parameter that is called name that will hold the name of the human. E.g.

function Human(name) {  
 this.setName(name);  
}

Properties:

* + \_name – a string (only letters and whitespaces)

1. Principal – Principal is module, **derived** from **Human** (extends Human) and should have:

Constructor:

* + name parameter – for its base module.

Properties:

* + All parent properties

1. Teacher – Teacher is module, **derived** from **Human** (extends Human) and should have:

Constructor:

* + name parameter – for it’s base module
  + teachingSubject parameter – should set the teacher’s teaching subject. Could be null (if we have a class teacher for example)

Properties:

* + All parent properties
  + \_teachingSubject – a Subject (should be one of the subjects given by Subject.getSubjects()), could be null

Methods:

* + addGradeToStudent(student, gradeParams) – adds a grade to a student.
    - student – a Student (only Student instances allowed)
    - gradeParams – an object with the following properties:
      * subject – A Subject (should be one of the subjects given by Subject.getSubjects()).   
         If the teacher’s \_teachingSubject is null, this property is obligatory because the system wouldn’t know for what subject the grade is.   
         Otherwise if null, the system will use the \_teachingSubject property.   
         Even if \_teachingSubject of the teacher isn’t null, you can still pass this parameter and a grade to the given subject will be added, not the teacher’s teaching subject.
      * mark – number
      * semester – a Semester (only Semester instances allowed)

Example:   
classTeacher.addGradeToStudent(goshko, {**mark**: 5, **subject**: availableSubjects.**MATH**, **semester**: semester1});  
  
geographyTeacher.addGradeToStudent(martina, {**mark**: 6, **semester**: semester2});

1. Student – Student is module, **derived** from **Human** (extends Human) and should have:

Constructor:

* + name parameter – for it’s base module

Properties:

* + All parent properties
  + \_grades – a list of grades (only instances of Grade allowed)

Methods:

* + addGrade(grade) – adds a grade to the student’s grades.
    - grade – a Grade (only instances of Grade allowed)
  + getGrades() – gets the student’s grades.

1. Subject – Subject is a static module, which holds all of the available subjects  
   Methods:
   * getSubjects() – returns an array of available subjects
   * isValidSubject(subject) – returns if a given subject by string is valid (is present in the available subjects array)
2. Semester – Semester is a module and should contain:  
   Constructor:
   * \_name – a string (only letters and whitespaces)

Properties:

* + \_name – A string (only letters and whitespace)

1. **Grade –** Grade is a module, which should have:

Constructor:

* + **mark** – the mark of the grade
  + **subject** – the grade’s subject
  + **semester** – the grade’s semester

Properties:

* + **\_mark** – Number
  + **\_subject** –a Subject (should be one of the subjects given by Subject.getSubjects())
  + **\_semester** – a Semester (only Semester instances allowed)

**All hidden module properties should have visible (not hidden) methods – getProperty and setProperty (validates the given value) attached to the modules prototype!**

Separate all models in different files. Expose constructors for creating instances of current model in the global object “**app**”.

You can see how the objects are created in the generator.js file for example.

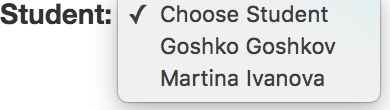
84 + 16 (unit tests) points

## Bonus: Implement showing grades

In the app.js file there is a TODO where you need to dynamically create the HTML option tags for the HTML select tag containing all students. You are allowed to use only Native JS (no external libraries such as jQuery). Think of a way to optimize the appending of elements.

**Keep in mind that if you do not do this part you will not be able to visualize the test data given with the skeleton project!**

Bonus 5 points



If you implement correctly all types, the code will work and will show something like this in the browser:

