UNIVERSITÀ DI PISA

Università degli Studi di Pisa

Laurea Specialistica in Informatica

Advanced Programming

Final Term Paper

Start Date: 3/06/2016
Submission deadline: 24/06/2016 (send a single PDF file to attardi@di.unipi.it)

Rules:

The paper must be produced personally by the student, signed implicitly via his mail address.

You are allowed to discuss with others the general lines of the problems, provided that each student eventually formulates his own solution. Each student is expected to understand and to be able to explain his solution.

You are allowed to consult documentation from any source, provided that references are mentioned.

It is not considered acceptable:

- **to consult or setup an online forum, to request help of consultants in producing the paper**
- to develop code or pseudo-code with others
- to use code written by others
- to let others use someone"s code
- to show or to examine the work of other students.

Violation of these rules will result in the cancellation of the exam and a report to the Presidente del Consiglio di Corso di Studio.

For the programming exercises you can choose a programming language among C++, C#, Java and JavaScript.

The paper must:

1. be in a single PDF file, formatted readably (**font size ≥ 10 pt** with suitable margins, single column), of **no more than 10 numbered pages**, including code: for each extra page one point will be subtracted from the score.
2. include the student name
3. provide the solution and the code for each exercise separately, referring to the code of other exercises if necessary.
4. cite references to literature or Web pages from where information was taken.

Introduction

You will have to implement a library of Web Componnts similar to React.JS, which maintains its internal Virtual DOM representation. The Virtual DOM is defined by this abstract JSON syntax:

```
Node = {
  'tag': string
  [, 'attrs': { [ string: string, ]* } ]
  [, 'children': null | [ string | Node ]* ]
}
```

i.e. a `Node` has a property named '`tag`' and optional properties '`attrs`' and '`children`'. `Attrs` are set of key/value pairs. `Children` are a sequence of either strings or `Node`s.

`React` components are defined with the following syntax:

```
React.class({
   constructor: <function>
   <props>
})

<props> = | , <name>: <value> <props>
```

The class should "inherit" from a predefined `React.Component` class. The `constructor()` method is invoked when the operator `new` is invoked on the component with the given arguments. A `render()` method should be provided to override the default one provided in class `Component` and it is expected to return a `Node` object. The library should provide a function:

```
React.render(Node, DOM);
```

which takes as arguments a `Node` and a DOM element and replace the DOM element with the result of rendering the `Node` according to its `render()` method.

Here is an example of a counter component:

```
var counter = React.class({
    constructor: function() { this.count = 0; }
    onClick: function(event) { this.count += 1; }
    render() {
        return [ { tag: 'span', children: this.count },
                 { tag: 'button',
                  attrs: { onClick: this.onClick },
                  children: 'Increment' } ]
    }
});
```

Exercise 1

Describe the choice of object model used for representing the `Component` classes and provide an implementation of the `React.Class` function and a basic implementation of `React.render` which modifies the `innerHTML` property of the DOM element.

Exercise 2

Extend the React components so that they keep the version of the `Node` at the last previous render invocation. Develop an optimized `React.render()` function which modifies only the parts of the DOM element that have been changed since the previous rendering.

Exercise 3

Devise a solution for connecting DOM events to the invocation of methods in a `Component` instance.

Test the solution on the example in the introduction and provide a JFiddle for testing it.

Exercise 4

Introduce the ability to express components in a style like JSX, i.e. using the following syntax:

```
jsxel = tagOpen jsxel* tagClose | <tag /> | jsxval
tagOpen = <tag [name=jsxval]* >
jsxval = string | { JS }
```

Design classes for representing the abstract syntax for JSX and implement a recursive descent parser that produces such representation reading from an input stream. Implement a code generator that turns this representation into a JavaScript representaion of a Node.

Exercise 5

Introduce briefly the event driven programming paradigm and discuss the benefits of asynchronous programming. List a few libraries or frameworks that support event driven programming. Provide a coding example to compare a solution based on multithreading and one based on asynchronous IO.