Programming Technology 2 - Academic Year 2020/21

Convocatoria extraordinaria (09/07/2021) - Duration: **2 hours**
Maximum grade: **8 points**
Minimum for passing the course: **4 points**

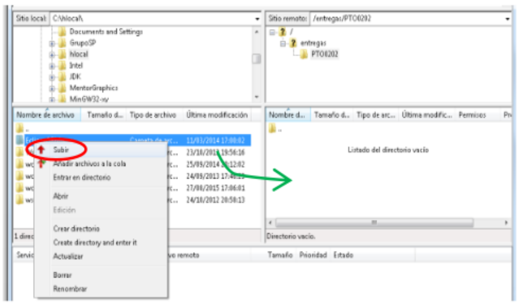
(Groups A,B,C,D,E,F,I and DG)

INSTRUCTIONS

---

1. **Download your assignment 2 from the Campus Virtual**, import it into Eclipse, and make sure it works correctly.
2. **Unmount drive Z** by clicking on the icon **"Desmontar Z"** that appears on the desktop.
3. Create a file called **ChangesExam.txt** inside the directory **src**. In this file you should include your full name, and for each question the name of the **.java** files that you have created or modified. You can add other comments about your solution as well.
4. **The submitted code must compile**, and should not break class encapsulation (e.g., accessing fields of other classes directly, etc). If your code does not compile or breaks the encapsulation, the grade of the corresponding question will be `0`**.**
5. When marking your assignment, we will take into account the functionality, the clarity, and the correct use of Object Oriented Programming.

HOW TO SUBMIT



To submit the exam, create a file **YourName.zip** that includes your project (without the bin directory). Double click on the icon **"EXAMENES en LABs entregas"** that appears on the desktop, this will open a new window. Inside the window click on **"ALUMNOS entrega de practicas y examenes"**. A new window will pop up, in which you have to select the zip file that you have created and drag it to the right panel (or use the right button of the mouse and then select the option **Subir**) as indicated in the Figure. Before leaving the lab, you have to pass by the professor's table to verify that your code has been submitted correctly and sign the submission form.

Question 1 [3 points]

In this question, you can create new classes and slightly modify class Main to add support for the new kind of Body, but you cannot modify any other class. Your assignment 2 should keep working correctly in both `GUI` and `batch` modes.

Implement a class **TempSensBody** that represents a body whose behaviour is affected by its temperatures as explained below. Its constructor receives the following parameters

```
public TempSensBody(String id, Vector2D v, Vector2D p, double m,
                    double minT, double maxT, double redF, double incF)
```

where the values of **`minT`** (minimum temperature) and **`maxT`** (maximum temperature) are positive, and the values of **`redF`** (temperature reduction factor) and **`incF`** (temperature increase factor) are between `0` and 1 (both exclusive). The constructor should throw a corresponding exception if any of these values is invalid.

This kind of body can be in two modes: ***moving*** or ***cooling***. Initially it is in a ***moving*** mode and its temperature is `0`. It moves according to the following behaviour:

1. If it is in a ***moving*** mode, then
   a. It first moves like a normal body.
   b. It adds **`incF*p1.distanceTo(p2)`** degrees to its temperature, where **`p1`** is the position after moving and **`p2`** is its position before moving.
   c. If the new temperature exceeds the value of **`maxT`** it changes to ***cooling*** mode.
2. If it is in a ***cooling*** mode, then it does not move at all but rather
   a. Its temperature is reduced to **`currentTemperature*(1-redF)`**
   b. If the new temperature is smaller than **`minT`** it changes to ***moving*** mode.

You should also implement a corresponding builder, and integrate it in the simulator, that accepts the following `JSON` structure

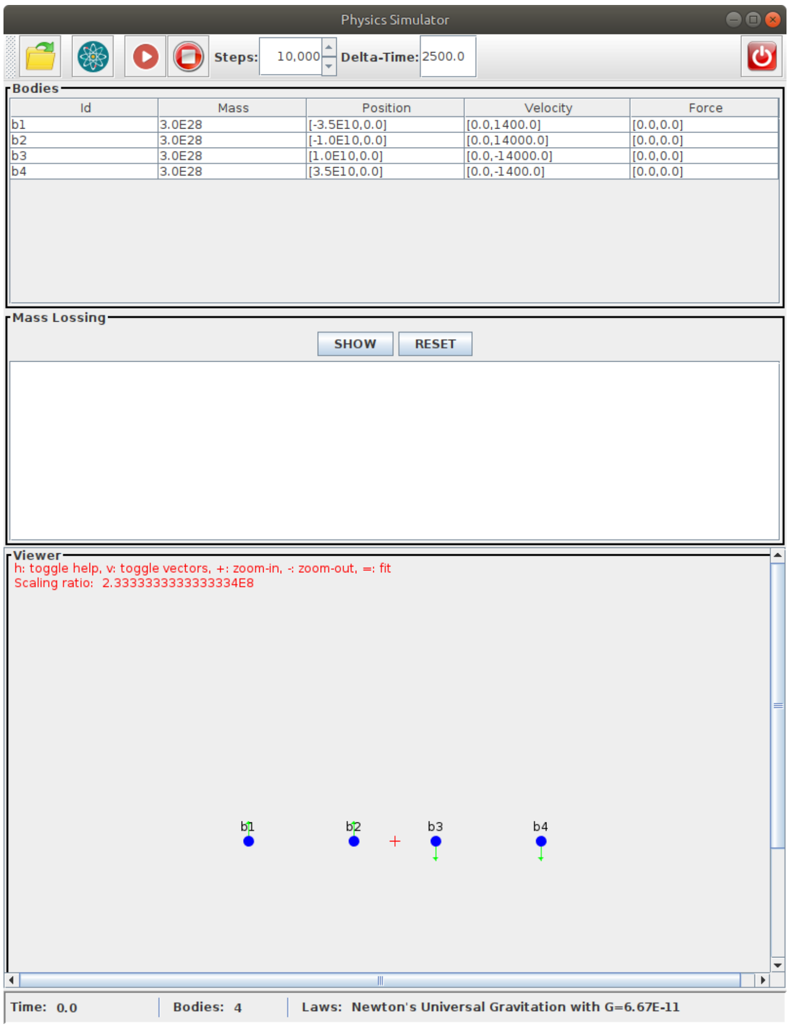
```
{
   "type" : "tempsens",
   "data" : {
              "id" : "b1",
              "p"  : [0.0e00, 4.5e10],
              "v"  : [1.0e04, 0.0e00],
              "m" : 1.5e30
             "minT" : 1e5,
             "maxT" : 10e5,
             "redF" : 0.3,
             "incF" : 10e-5
           }
 }
```

where the parameters **`minT`**, **`maxT`**, **`redF`** and **`incF`** are optional with default values **`1e5`**, **`10e5`**, **`0.3`**, and **`10e-5`** respectively.

Question 2 [5 points]

In this question, you can only modify/add classes in the package view, i.e., you're not allowed to touch the model or the controller. You cannot use casting, getClass, or instanceOf, etc. Solutions that do not meet these requirements will be graded with `0`.

Add to main window a new panel that includes two buttons and a text area, as in the following figure (where we just started the simulator with an input file):



Id	Mass	Position	Velocity	Force
b1	3.0E28	[-3.5E10,0.0]	[0.0,1400.0]	[0.0,0.0]
b2	3.0E28	[-1.0E10,0.0]	[0.0,14000.0]	[0.0,0.0]
b3	3.0E28	[1.0E10,0.0]	[0.0,-14000.0]	[0.0,0.0]
b4	3.0E28	[3.5E10,0.0]	[0.0,-1400.0]	[0.0,0.0]

```
h: toggle help, v: toggle vectors, +: zoom-in, -: zoom-out, =: fit
Scaling ratio:  2.3333333333333334E8
```

```
Time:  0.0    Bodies:  4    Laws:  Newton's Universal Gravitation with G=6.67E-11
```

When the simulation is running, the text area should always be empty. When the simulation is not running, if the user presses the **`SHOW`** button it should show in the text area, for each body, a line with its identifier and the total mass that it has lost so far. Next we explain the meaning of the total mass loss.

The mass loss is accumulated during the execution, i.e., if a body **b** loses 5 mass units in one execution step, and then loses 3 mass units in the next execution step, then pressing **`SHOW`** the text area will include the following line for that body

```
b          8.0
```

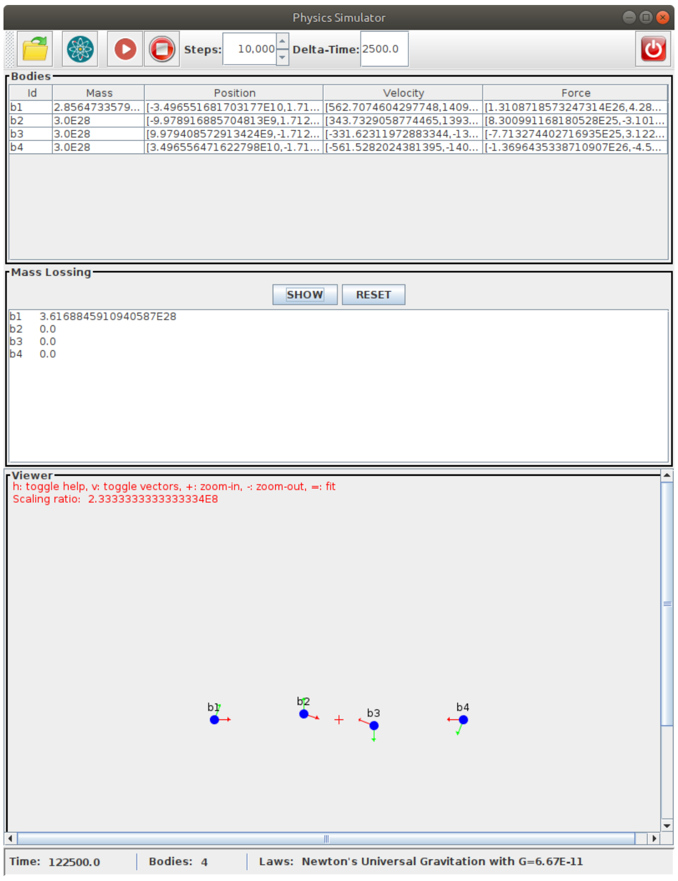
The **RESET** button allows resetting the mass loss to `0.0`, i.e., we start to count the mass loss from that moment on, and, in addition it shows the current mass loss (which is `0`) for each body in the text area. For example, in the example above, if after the first step in which **b** loses 5 mass units we press **RESET**, then the text area will include the following line for the corresponding body:

```
b          0.0
```

and if we execute the simulation one more step in which it loses 3 mass units, then pressing **SHOW** the text area will include the following line for the corresponding body:

```
b          3.0
```

As another example, the window on the left shows the result when pressing **SHOW** after some execution steps. Note that only **b1** has lost mass. The window on the right is the result after pressing **RESET**.

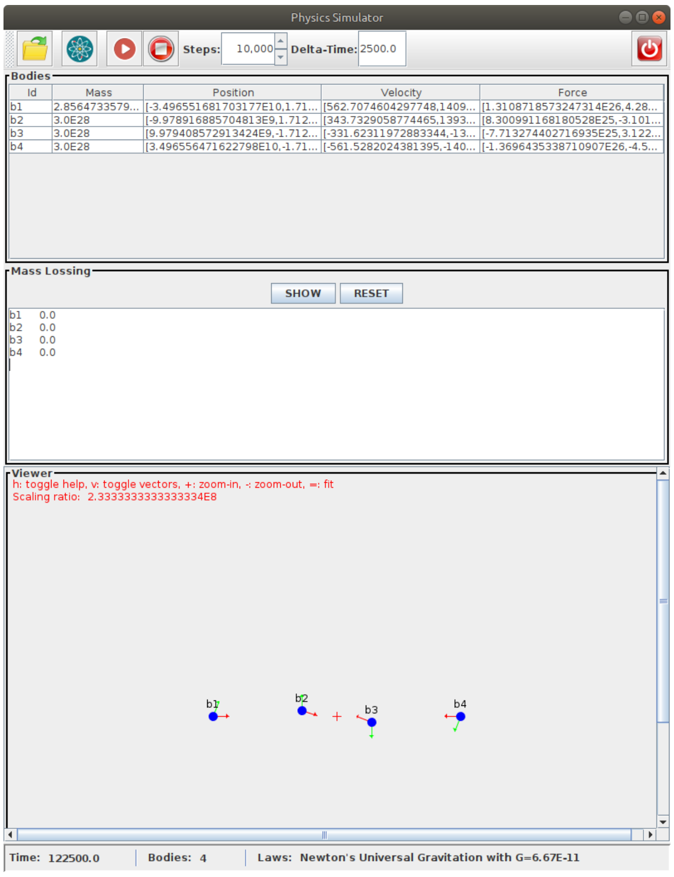


Id	Mass	Position	Velocity	Force
b1	2.8564733579...	[-3.496551681703177E10,1.71...	[562.7074604297748,1409...	[1.310871857324731 4E26,4.28...
b2	3.0E28	[-9.978916885704813E9,1.712...	[343.7329058774465,1393...	[8.300991168180528E25,-3.101...
b3	3.0E28	[9.979408572913424E9,-1.712...	[-331.62311972883344,-13...	[-7.713274402716935E25,3.122...
b4	3.0E28	[3.496556471622798E10,-1.71...	[-561.5282024381395,-140...	[-1.3696435338710907E26,-4.5...

```
b1    3.6168845910940587E28
b2    0.0
b3    0.0
b4    0.0
```

h: toggle help, v: toggle vectors, +: zoom-in, -: zoom-out, =: fit
Scaling ratio: 2.3333333333333334E8

Time: 122500.0   Bodies: 4   Laws: Newton's Universal Gravitation with G=6.67E-11



Id	Mass	Position	Velocity	Force
b1	2.8564733579...	[-3.496551681703177E10,1.71...	[562.7074604297748,1409...	[1.310871857324731 4E26,4.28...
b2	3.0E28	[-9.978916885704813E9,1.712...	[343.7329058774465,1393...	[8.300991168180528E25,-3.101...
b3	3.0E28	[9.979408572913424E9,-1.712...	[-331.62311972883344,-13...	[-7.713274402716935E25,3.122...
b4	3.0E28	[3.496556471622798E10,-1.71...	[-561.5282024381395,-140...	[-1.3696435338710907E26,-4.5...

```
b1    0.0
b2    0.0
b3    0.0
b4    0.0
```

h: toggle help, v: toggle vectors, +: zoom-in, -: zoom-out, =: fit
Scaling ratio: 2.3333333333333334E8

Time: 122500.0   Bodies: 4   Laws: Newton's Universal Gravitation with G=6.67E-11