Departamento de Informática
Universidad Carlos III de Madrid

Bachelor's Degree in Computer Science

Programming Final Exam

December 2019

READ CAREFULLY THESE INSTRUCTIONS BEFORE STARTING THE EXAM, NOT FOLLOWING THEM WILL BE PENALIZED WITH AT LEAST 1 POINT

- You have **3 hours**.
- Do not answer on this sheet, use the provided ones.
- This is your personal copy, keep it after the exam. Use it as you want.
- Fill in all the pages with a pen (personal data and answers). Do not use a pencil or a red pen.
- Do not forget to include your name and surname on every page.
- Notes, slides, books, etc. are allowed. Electronic devices are not allowed.
- Problems can be solved in any order, but if so, use a new sheet for each problem and deliver them in the right order. Also, deliver the problem sections in order. It is recommended to use a sheet for each class.
- You are not permitted to leave the room during the exam unless you have handed it in.
- It is not allowed:
  - The use of built-in functions or libraries not seen during the course.
  - The use of `break` and `continue` statements.
- The only list built-in functions allowed are: append, insert and remove.
- Read through the entire exercise before you begin.

A Warehouse Management System

Retail companies like Amazon are experimenting with the use of drones to deliver goods to their clients. This will reduce the delivery costs and the time needed to do it. The purpose of this exercise is to simulate a warehouse management system, where drones are used, according to the following description:

- Each warehouse has a name, a series of drones, serves a given neighborhood and is able to receive orders.
- A drone has an id, a current power (in the range 0 to 100), a status (available or busy) and can be used to deliver an order (initially they have no order).
- An order has a priority, an id and the address where it has to be delivered.
- The map of the area has a name of the neighborhood and is a matrix where each element is either a string representing the id of a house or 's' for positions where there are no houses (see example below)
- For every order in the warehouse, a drone is selected to serve it. See below details about how drones are selected.

Considering the above description, you are asked to:

1. (1pt) Define the needed classes and their `init` methods creating properties for all the attributes needing it. Creating non-needed properties will be penalized.
2. (0.5pt) Create the method `createDrones(self,Num)` that returns a list of `Num` drones with consecutive ids `drone1`, `drone2`, ..., `droneNum`. Indicate the class where this method belongs to.
3. (2pt) Create the method `createAreaMap(self,whId,rows,cols,NumH)` that given the name of the warehouse (`whId`), the dimension of the area in terms of number of rows (`rows`) and number of columns (`cols`), and the number of houses in the area (`NumH`), returns a `rows`×`cols` matrix represented as a list of lists, which stores strings to represent the elements in the map. The string `"s"` will represent streets with no client's houses on

them; the string `whId` will represent the warehouse position; and the strings `"h1"` to `"hNumH"` will represent the houses ids. The position for each element in the map will be selected randomly. Get sure all the houses are created and house addresses are not repeated. Indicate the class where this method belongs to.

Example of a 4×5 map area with 10 houses for the warehouse `wh0`:

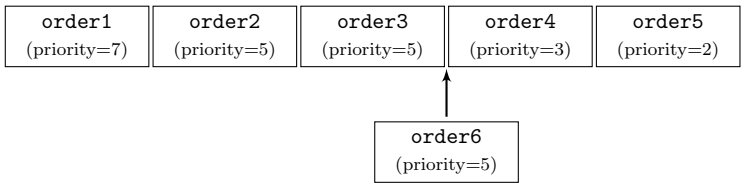
s	h10	s	h7	h8
s	s	s	h9	wh0
s	s	h3	h2	h4
h5	s	s	h1	h6

4. (2pt) Create the method `computeDistance(self,whId,houseId)` that given the warehouse id (`whId`) and a house id (`houseId`), returns the Euclidean distance (or the straight-line distance) between the two of them. The Euclidean distance is computed using the following formula: $d = \sqrt{(r_o - r_w)^2 + (c_o - c_w)^2}$, where r_o and c_o refer to the row and the column where the address of the house is located; and r_w and c_w refer to the row and the column where the warehouse is located. The square root of a number x can be computed as $x**0.5$. Indicate the class where this method belongs to.

   Examples: the Euclidean distance given the inputs `wh0` and `h5` is $d = \sqrt{(3-1)^2 + (0-4)^2}$; and the Euclidean distance given the inputs `wh0` and `h3` is $d = \sqrt{(2-1)^2 + (2-4)^2}$.

5. (1pt) Create the method `insertOrderInList(self,order)` that given an `order`, inserts it in the list of orders according to its priority value (higher priority orders first). If there are more than one order with same priority values, the current order will be inserted at the end of the orders with same priority. This method does not have a return statement. Indicate the class where this method belongs to.

   Example: assuming the current order `order06` with priority value 5:



6. (2pt) Create the method `pickDrone(self,orderAddr)` that given an order address `orderAddr`, returns the position in the list of drones of the warehouse of the drone serving it, and the distance from the warehouse to `orderAddr`. The chosen drone will be the first available one whose power is at least two times the distance from the warehouse to the orderAddr. If no such drone is found, the one with highest current power will be instantly fully recharged and selected. Indicate the class where this method belongs to.

7. (1pt) Create the method `processOrder(self,ord)` that inserts the order `ord` in the list of orders of the warehouse, picks the most suitable drone to process the order, and updates the pertinent attributes of the chosen drone. This method does not have a return statement. Indicate the class where this method belongs to.

8. (0.5pt) In a main program, ask the warehouse manager the number of drones that the warehouse counts with, the number of rows and the number of columns that covers the area, and the number of houses in the area. Then, create a warehouse; for such warehouse, create five orders to process with consecutive ids `order1`, `order2`, ..., `order5`.