Lesson 6. Introduction to object-oriented programming

Exercise 1. Develop a class named Matrix to represent a real-number matrix. The class must have three attributes: number of rows, number of columns, two-dimension array of double values.

Implement the following methods:

(NOTE: If the dimensions of the matrices passed as parameters to the methods are not correct to perform the corresponding operation, the method must return the special value `null`.)

a. Basic constructor

Parameters: `int rows, int cols, double a`, `double b`

Operation: Allocates memory for the values array and initializes rows and cols

b. Random initialization

Parameters: `double a`, `double b`

Returns: Nothing

Operation:

Initializes the matrix with random values in the range `[a, b)`

c. Read matrix values

Parameters: None

Returns: Nothing

Operation:

Initializes the matrix with values read from the keyboard

d. Print matrix

Parameters: None

Returns: Nothing

Operation: Prints matrix on the screen

e. Find the maximum value of the matrix

Parameters: None

Returns: `double max`

Operation:

Retrieves max value of the array

f. Addition

Parameters: `Matrix m`

Returns: `Matrix r`

Operation:

`r = matrix + m`

g. Subtraction

Parameters: `Matrix m`

Returns: `Matrix r`

Operation:

`r = matrix - m`

h. Scalar multiplication

Parameters: `double x`

Returns: `Matrix r`

Operation:

`r = x` x `matrix`

i. Multiplication

Parameters: `Matrix m`

Returns: `Matrix r`

Operation: `r = matrix x m`

j. Transpose

Parameters: None

Returns: `Matrix r`

Operation:

`r = transpose(matrix)`

Develop an auxiliary class to test the functioning of the `Matrix` class. The `main` method of this class must create two matrices `m1` and `m2`. The values of `m1` are read from the keyboard; the values of `m1` are randomly initialized –the number of rows and columns of both matrices are read from the keyboard. Print `m1` and `m2`. Calculate `m3 = transpose(transpose(m1)+m2)` and print `m3`.

Change the visibility of the attributes to `private` and extend conveniently the implementation.

Exercise 2. Develop the following classes:

Point class

Attributes

`double x`: *x* coordinate

`double y`: *y* coordinate

Constructors

  - `Point(double x, double y)`: initialize x, y coordinates
  - `Point()`: initialize x, y coordinates to (0, 0)
- Methods
  - `void setX(double x)`: assigns x coordinate
  - `void setY(double y)`: assigns y coordinate
  - `double getX()`: get x coordinate
  - `double getY()`: get y coordinate
  - `double distance(Point p)`: distance between the point and p
  - `void print()`: print point data

Rectangle class

- Attributes
  - *a* point (upper-left)
  - *b* point (bottom-right)
- Constructors
  - `Rectangle(Point a, Point b)`: initialize a, b points
- Methods
  - `void setA(Point a)` : assigns upper-left point
  - `void setB(Point b)` : assigns bottom-right point
  - `Point getA()` : get upper-left point
  - `Point getB()` : get bottom-right point
  - `double height()` : returns the height of the rectangle
  - `double width()` : returns the width of the rectangle
  - `double area()` : returns the area of the rectangle
  - `void move(double dx, double dy)` : moves the rectangle
  - `boolean inside(Rectangle r)` : true if the rectangle is inside `r`
  - `void print()` : print rectangle data

Develop an auxiliary class to test the use of *Point* and *Rectangle*. The `main` method of the class must create two rectangles –the coordinates of the corners must be read from the keyboard– and print on the screen if one of the rectangles is inside the other.

Extend the program to perform the same operation with 10 rectangles.

Exercise 3. Develop the following classes:

Tyre class

- Attributes
  - `double r`: radius
  - `String type`: “Dry”, “Wet”, etc.
- Constructors
  - `Tyre(double r, String type)`: initialize tyre with the given values
  - `Tyre()` : initialize a tyre (radius = 1, type = “Dry”)
- Methods
  - get/set methods for the attributes
  - `void print()`: prints a string with tyre information

Chassis class

- Attributes

`double w`: weight

`String material`: "Aluminum", "Iron", etc.

Constructors

`Chassis(double w, String material)`: initialize chassis with the given values

`Chassis()`: initialize a chassis (weight = 1000, material = "Aluminum")

Methods

get/set methods for the attributes

`void print()`: prints a string with chassis information

Car class

Attributes

`Tyre [] ty`: 4-size tyre array

`Chassis ch`: chassis

`String c`: car color

Constructors

`Car(Tyre t, Chassis ch, String c)`: initialize chassis with the given values (create 4 copies of `t` to be assigned as the tyres of the car)

Methods

`void print()`: prints a string with car information

`void setColor(String c)`: set car color

`void setTyre(Tyre t, int pos)`: set a tyre in the position

`void setChassis(Chassis c)`: set chassis

ManufacturingPlant class

Attributes

`double defaultRadius`: default tyre radius

`String defaultType`: default tyre type

`double defaultWeight`: default chassis weight

`String defaultMaterial`: default chassis material

`String [] colors`: allowed colors

Constructors

No constructors (use default constructor)

Methods

get/set methods for the default manufacturing parameters

`Car manufactureCar()`: the method creates a flawless car with probability 99% (use `Math.random()`). If the process 'fails', return `null`. The color of the car is randomly chosen from the colors array

Create an auxiliary class to test the classes. The `main` method of the class must create two manufacturing plants. Each plant must manufacture 100 cars –if the *manufactureCar* process fails, a new car must be manufactured. Print on the screen the information of the new cars and the actual flaw rate of each plant.

Programming – Grado en Ingeniería Informática

Authors

Of the English version:
Juan Gómez Romero

Based on the work by:
Ángel García Olaya
Manuel Pereira González
Silvia Castro García
Gustavo Fernández-Baillo Cañas
Daniel Pérez Pinillos
Javier Ortiz Laguna
Álvaro Torralba Arias de Reyna

Universidad
Carlos III de Madrid