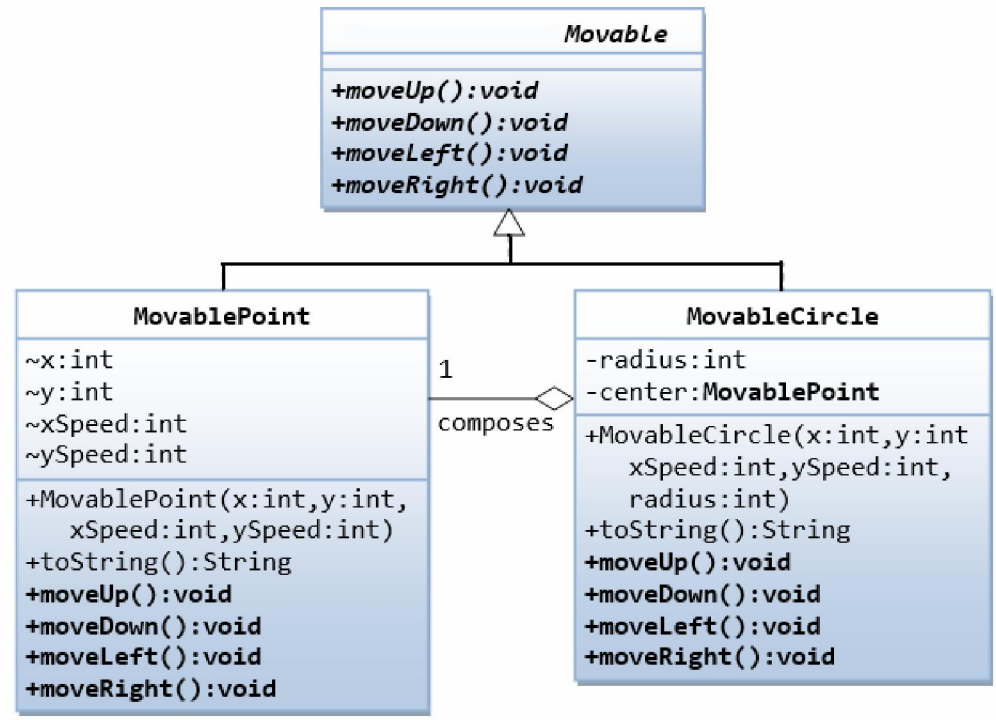
Exercițiul 13

Se dau clasele `Movable`, `MovablePoint` şi `MovableCircle`, unde `MovablePoint` şi `MovablecCircle` sunt derivate din `Movable` (a se vedea diagrama de clase UML).

Clasa `Movable` este o clasă abstractă. Clasa `MovablecCircle` va avea o dată membră de tip `MovablePoint`.



Clasa `MovablePoint` va conține următoarele elemente (a se vedea diagrama UML a clasei):

- două atribute private, `x` şi `y` de tipul întreg, şi care au valoarea implicită `0`.
- două atribute private, `xSpeed` şi `ySpeed` de tipul întreg, şi care au valoarea implicită `0`.
- constructor cu parametri, constructor de copiere, metode *getter* şi *setter* pentru atributele private.
- o metodă `setXY()` pentru a iniția liza ambele coordonate `x` şi `y` ale unui punct.
- metodele `moveUp()`, `moveDown()`, `moveRight()`, `moveLeft()` ce contribuie la deplasarea unui punct în sus, în jos, la stânga şi la dreapta.
- o metodă `print()` ce afişează "`Punct[(x,y), viteza=(dx, dy)]`".

- La fel ca la multe dintre exemplele anterioare, se doreşte să se specifice declaraţia clasei în fişierul `MovablePoint.h` iar definiţia metodelor clasei să fie realizată în fişierul `MovablePoint.cpp`.

Clasa `MovablePoint` va avea următoarea declaraţie:

```
class MovablePoint : public Movable {
  private:
    int x, y;
    int xSpeed, ySpeed;

  public:
    MovablePoint(int x = 0, int y = 0, int xSpeed = 0, int ySpeed = 0);
    MovablePoint(const MovablePoint&);

    int getX() const;
    void setX(int);

    int getY() const;
    void setY(int);

    int getXSpeed() const;
    void setXSpeed(int);

    int getYSpeed() const;
    void setYSpeed(int);

    void setXY(int, int);

    void moveUp();
    void moveDown();
    void moveLeft();
    void moveRight();

    void print();
};
```

Clasa `MovableCircle` va avea următoarea structură:

- două atribute private, `center` de tipul `MovablePoint`, şi `radius` de tip `int` şi care are valoarea implicită `1.0`.
- constructor cu parametri, constructor de copiere.
- metode *getter* şi *setter* pentru atributul `radius`.
- Operatorul '=' supraîncărcat.
- Metodele `moveUp()`, `moveDown()`, `moveRight()`, `moveLeft()` ce contribuie la deplasarea cercului în sus, jos, la stânga şi la dreapta.
- o metodă `print()` ce afişează "`Cerc[Punct[(x, y), viteza=(dx, dy)], raza=X]`".

Clasa `MovableCircle` va avea următoarea declaraţie:

```
class MovableCircle : public Movable {
  private:
    int radius;
```

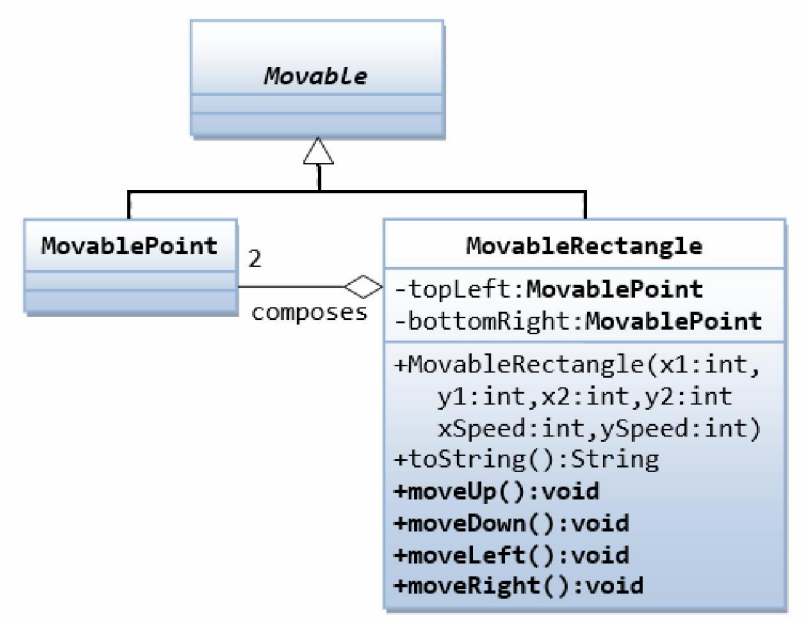
```
    MovablePoint center;
  public:
    MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius = 1);

    MovableCircle(const MovableCircle&);

    MovableCircle& operator= (const MovableCircle&);

    int getRadius() const;
    void setRadius(int);

    void moveUp();
    void moveDown();
    void moveLeft();
    void moveRight();

    void print();
};
```

Referințe:

Clasele `MovablePoint` şi `MovableCircle`.

To do

Să se declare şi să se definească clasa `MovableRectangle` (a se vedea diagrama de clase UML).



Clasa `MovableRectangle` va avea următoarea structură:

- două atribute private, `topLeft` şi `bottomRight` de tip `MovablePoint`.
- constructor cu parametri, constructor de copiere.
- Operatorul '=' supraîncărcat.
- Metodele `moveUp()`, `moveDown()`, `moveRight()`, `moveLeft()` ce contribuie la deplasarea dreptunghiului în sus, în jos, la stânga şi la dreapta.

- o metodă `print()` ce afişează "`Dreptunghi[Punct[(x1, y1), viteza=(dx1, dy1)], Punct[(x2, y2), viteza=(dx2, dy2)]]`".

Conţinutul fişierului `test-rectangle.cpp` ar putea fi:

```
#include <iostream>
#include <typeinfo>
#include "MovableRectangle.h"

int main() {
  Movable* m1 = new MovableRectangle(3, 7, 9, 11, 5, 3);

  cout << "Initial obiectul de tip "
       << typeid(*m1).name()
       << " are valoarea: " << endl;
  m1->print();
  cout << endl;

  int time = 0;
  while (time < 10) {
    time++;
    switch (time % 4) {
      case 0: m1->moveUp();
              break;
      case 1: m1->moveRight();
              break;
      case 2: m1->moveDown();
              break;
      case 3: m1->moveLeft();
              break;
    }

    cout << "Dupa " << time << "s: ";
    m1->print();
    cout << endl;
  }

  return  0;
}
```