Tema 3

Termen de predare: laboratorul din săptămâna a VII-a.

Obiective:

- Relaţia de *moştenire* (IS-A)
- *Early binding* şi *late binding*
- Metode virtuale
- Metode virtuale pure
- Design patterns: composite, factory method, singleton
- Polimorfism static: parametrizări de clase
- Excepţii

Bibliografie:

- The Liskov Substitution Principle : http://www.objectmentor.com/resources/articles/lsp.pdf
- http://www.codeproject.com/Articles/386982/Two-Ways-to-Realise-the-Composite-Pattern-in-Cplus
- http://www.codeproject.com/Articles/363338/Factory-Pattern-in-Cplusplus

Cerinţe:

- Problemele trebuie prezentate până în săptămâna a 7-a inclusiv
- Fiecare problemă este notată cu maxim 10 puncte
- Toate problemele sunt obligatorii

Probleme:

1. Implementați o ierarhie de clase care reprezintă figuri geometrice.

```
#ifndef PUNCT_H__
#define PUNCT_H__

#include <istream>
#include <ostream>

class Punct
{
 private:
  int x, y;

 public:

  Punct(int = 0, int = 0);
  int getX() const;
  int getY() const;
  void setX(int);
  void setY(int);
  void moveBy(int, int);

  friend std::istream &operator>>(std::istream &, Punct &);
  friend std::ostream &operator<<(std::ostream &, const Punct &);
};

#endif
#ifndef SHAPE_H__
#define SHAPE_H__

#include <ostream>
#include <istream>
```

```
class Shape
{
 public:
  virtual void output(std::ostream &) const = 0;
  virtual void moveBy(int, int) = 0;
  virtual void readFrom(std::istream &) = 0;
  virtual ~Shape() = 0;
  friend std::ostream &operator<<(std::ostream &, const Shape &);
};

#endif
#ifndef CERC_H__
#define CERC_H__

#include <ostream>
#include <istream>
#include <string>
#include "shape.h"
#include "punct.h"

class Cerc : public Shape
{
 private:
  Punct c;
  int r;

 public:
  static const std::string identifier;

  Cerc(const Punct & = Punct(), const int = 0);
  ~Cerc();
  void output(std::ostream &) const;
  void readFrom(std::istream &);
  void moveBy(int, int);

  friend std::ostream &operator<<(std::ostream &, const Cerc &);
};

#endif
#ifndef DREPTUNGHI_H__
#define DREPTUNGHI_H__

#include <ostream>
#include <istream>
#include <string>
#include "shape.h"
#include "punct.h"

class Dreptunghi : public Shape
{
 private:
  Punct p1, p2;

 public:
  static const std::string identifier;

  Dreptunghi(const Punct & = Punct(), const Punct & = Punct());
  ~Dreptunghi();
  void output(std::ostream &) const;
  void readFrom(std::istream &);
  void moveBy(int, int);

  friend std::ostream &operator<<(std::ostream &, const Dreptunghi &);
};

#endif
#ifndef TRIUNGHI_H__
#define TRIUNGHI_H__

#include <ostream>
#include <string>
#include "shape.h"
#include "punct.h"

class Triunghi : public Shape
{
 private:
  Punct p1, p2, p3;

 public:
```

```
  static const std::string identifier;

  Triunghi(const Punct & = Punct(), const Punct & = Punct(), const Punct & = Punct());
  ~Triunghi();
  void output(std::ostream &) const;
  void readFrom(std::istream &);
  void moveBy(int, int);

  friend std::ostream &operator<<(std::ostream &, const Triunghi &);
};

#endif
```

2. Citiți despre șablonul de proiectare *Composite* (e.g., aici: http://www.codeproject.com/Articles/386982/Two-Ways-to-Realise-the-Composite-Pattern-in-Cplus). Implementați o clasă *Grup* care să reprezinte o figură geometrică alcăuită din alte figuri geometrice.

```
#ifndef GRUP_H__
#define GRUP_H__

#include <ostream>
#include <istream>
#include <string>
#include <vector>
#include "shape.h"
#include "punct.h"

class Grup : public Shape
{
 private:
  std::vector<Shape *> continut;

 public:
  static const std::string identifier;

  Grup();
  ~Grup();
  void add(Shape *);
  void remove(Shape *);
  void output(std::ostream &) const;
  void readFrom(std::istream &);
  void moveBy(int, int);

  friend std::ostream &operator<<(std::ostream &, const Grup &);
};

#endif
```

3. Citiți despre implementarea *factory*-urilor în C++ (e.g., aici: http://www.codeproject.com/Articles/363338/Factory-Pattern-in-Cplusplus). Implementați clasa *ShapeFactory*, care reprezintă un factory cu ajutorul căruia se pot construi *Shape*-uri.

```
#ifndef SHAPEFACTORY_H__
#define SHAPEFACTORY_H__

#include <istream>
#include <map>
#include <string>
#include "shape.h"

typedef Shape *(createShapeFunction)(void);

/* thrown when a shape cannot be read from a stream */
class WrongFormatException { };

class ShapeFactory
{
 public:
  static void registerFunction(const std::string &, const createShapeFunction *);
  static Shape *createShape(const std::string &);
  static Shape *createShape(std::istream &);

 private:
  std::map<std::string, createShapeFunction *> creationFunctions;
  ShapeFactory();
```

```
  static ShapeFactory *getShapeFactory();
};

#endif
```

Asigurați-vă că toate clasele pe care le-ați scris funcționează corect, testându-le folosind următorul program principal. Care este diferența dintre obiectele inițial create de program (aflate în vectorul `shapes`) și cele citite din fișier (aflate în vectorul `shapesRead`)?

```
#include <iostream>
#include <fstream>
#include "shape.h"
#include "triunghi.h"
#include "dreptunghi.h"
#include "cerc.h"
#include "grup.h"
#include "shapefactory.h"

using namespace std;

vector<Shape *> shapes;

void createSampleShapeVector()
{
  cout << "Shapes created:" << endl;
  Triunghi *t1 = new Triunghi(Punct(10, 10), Punct(20, 20), Punct(10, 20));
  Triunghi *t2 = new Triunghi(Punct(15, 15), Punct(35, 35), Punct(35, 15));
  Shape &s1(*t1);
  Shape &s2(*t2);

  cout << s1 << endl;
  t1->moveBy(6, 7);
  cout << s1 << endl;
  s1.moveBy(1, 0);
  cout << s1 << endl;
  cout << s2 << endl;
  cout << endl;

  Dreptunghi *d1 = new Dreptunghi(Punct(15, 15), Punct(35, 35));
  Shape &s3(*d1);

  cout << s3 << endl;
  d1->moveBy(5, 5);
  cout << s3 << endl;
  s3.moveBy(-10, -10);
  cout << *d1 << endl;
  cout << endl;

  Cerc *c1 = new Cerc(Punct(5, 5), 10);
  Shape &s4(*c1);

  cout << s4 << endl;
  c1->moveBy(5, 5);
  cout << s4 << endl;
  s4.moveBy(-10, -10);
  cout << *c1 << endl;
  cout << endl;

  Grup *g1 = new Grup();
  Shape &s5(*g1);
  g1->add(&s1);
  g1->add(&s2);
  g1->add(&s3);
  g1->add(&s4);

  cout << s5 << endl;
  s1.moveBy(100, 100);
  cout << s5 << endl;
  cout << endl;

  Grup *g2 = new Grup();
  g2->add(g1);
  g2->add(new Cerc(Punct(3, 4), 5));
  cout << *g2 << endl;

  shapes.push_back(t1);
  shapes.push_back(t2);
  shapes.push_back(c1);
```

```
  shapes.push_back(d1);
  shapes.push_back(g1);
  shapes.push_back(g2);
}

int main()
{
  createSampleShapeVector();

  cout << endl << endl << "Writing shapes to file" << endl;
  ofstream out("shapes.txt");
  out << shapes.size() << endl;
  for (int i = 0; i < (int)shapes.size(); ++i) {
    out << *shapes[i] << endl;
  }
  out.close();

  cout << endl << endl << "Reading shapes from file" << endl;
  vector<Shape *> shapesRead;
  ifstream in("shapes.txt");
  int n;
  in >> n;
  for (int i = 0; i < n; ++i) {
    Shape *s = ShapeFactory::createShape(in);
    shapesRead.push_back(s);
  }
  in.close();

  cout << endl << endl << "Shapes read from file:" << endl;
  for (int i = 0; i < (int)shapesRead.size(); ++i) {
    cout << *shapesRead[i] << endl;
  }
  cout << endl;
}
```

4. Adăugaţi clasa `Patrat` în ierarhia de clase. Ce relaţie există între clasa `Patrat` şi clasa `Dreptunghi`?

5. Parametrizaţi ierarhia de clase definită anterior în funcţie de tipul coordonatelor folosite. Astfel, `Triunghi<int>` va reprezenta triunghiuri ale căror coordonate sunt numere întregi, în timp ce `Triunghi<double>` va reprezenta triunghiuri ale căror coordonate sunt numere reale.