Problema 6: La jerarquia de polígons regulars

Esteve Brugulat Josep M. Ribó

30 d'octubre de 2009

1 Objectius

- Introduir l'herència
- Introduir el polimorfisme

Per fer aquest problema, cal que us llegiu:
- Els apunts (apartats 1.12 i 1.13)

És molt important que compileu i executeu el codi d'aquest problema.

2 La classe `RegularPolygon`

Un polígon regular és una figura geomètrica tancada formada per un nombre d'arestes (o costats) més gran o igual a tres. Els polígons regulars es caracteritzen perquè tenen tots els costats de la mateixa longitud.

Com a exemples de polígons regulars tenim: el quadrat, el triangle, el pentàgon, la circumferència (considerarem també una circumferència com a polígon regular amb infinites arestes de longitud = 0).

Les operacions que voldrem que ens ofereixi aquesta classe són les següents:

- Constructora per defecte
- Constructora amb la longitud de l'aresta.
- Set/get la longitud de l'aresta
- Get l'àrea del polígon
- Get el perímetre del polígon
- Generar un MyString que contingui una descripció de les característiques del polígon:
  - El seu nom ("quadrat", triangle"...)
  - La longitud de l'aresta
  - El seu perímetre
  - La seva àrea
  - Alguna característica específica pròpia de cada tipus particular de polígon. Per exemple, en el cas del triangle voldrem descriure'l també per la seva alçada i a la circumferència, pel seu radi.

  Aquesta operació l'anomenarem `toString()`.

- Get l'alçada del polígon (només en el cas del triangle)
- Get el radi del polígon (només en el cas de la circumferència).

El color el codificarem mitjançant un enter.

1. Representa i implementa la classe `RegularPolygon`

   Per fer-ho, visita els **apartats 1.12, 1.13.1, 1.13.2, 1.13.3 i 1.13.4 dels apunts** i reflexiona sobre els punts següents:

   - Quins atributs cal posar en aquesta classe?
   - Quin codi tenen a la classe `RegularPolygon` les operacions `getArea, getPerimeter`?
   - I `toString`?
   - Has de posar a la classe `RegularPolygon` les operacions `getRadius` i `getHeight`?

   Fitxer `RegularPolygon.h`:

```
#include "MyString.h"

class RegularPolygon{

  double edgeLength;

public:
  RegularPolygon();
  RegularPolygon(int pcolor, int px, int py, double plength);
  void setEdgeLength(double plength);

  double getEdgeLength();
  virtual MyString toString();
  virtual double getArea()=0;
  virtual double getPerimeter()=0;

};
```

   Comentaris:

   - *Les operacions* `getArea` *i* `getPerimeter` *són* **virtuals pures** *ja que el càlcul de l'àrea/perímetre d'un polígon depén del tipus concret de polígon amb el qual treballem (e.g. àrea quadrat: costat*costat; àrea triangle: base*alçada/2; àrea circumferència: PI*radi*radi...).*
   - *L'operació* `toString` *és declarada virtual perquè haurà de ser redefinida en cada subclasse de* `RegularPolygon` *(cadascuna tindrà les seves pròpies característiques específiques, com el radi o l'alçada).*
   - *Com la classe* `RegularPolygon` *té operacions virtuals pures (*`getArea, getPerimeter`*), és una* **classe abstracta**. *Per tant, no podrem crear objectes directament d'aquesta classe:*
     `RegularPolygon p; //INCORRECTE`
   - *Encara que* `RegularPolygon` *sigui abstracta, sí que tindrà operacions constructores per tal d'inicialitzar els seus atributs. Aquestes operacions constructores seran cridades implícitament quan es crein objectes de les subclasses de* `RegularPolygon` *(e.g.,* `Square, Circumference...`*).*
   - *Les operacions* `getRadius` *i* `getHeight` *no s'han de posar a la classe* `RegularPolygon` *perquè no tenen sentit per a ella (hi ha polígons que no tenen radi o pels quals no té massa sentit definir l'alçada).*

Fitxer `RegularPolygon.cc`**:**

```
RegularPolygon::RegularPolygon()
  {
    edgeLength=0;
  }

RegularPolygon::RegularPolygon(double plength)
  {
    edgeLength=plength;
  }

void RegularPolygon::setEdgeLength(double plength)
{
  edgeLength=plength;
}

double RegularPolygon::getEdgeLength()
{
  return edgeLength;
}

MyString RegularPolygon::toString()
{
  return MyString("Edge length: ")+
         MyString(getEdgeLength())+
         MyString("Area: ")+MyString(getArea())+
         MyString("Perimeter: ")+MyString(getPerimeter());
}
```

Comentaris:

- *No hem d'implementar les operacions virtuals pures* `getArea` *i* `getPerimeter` *però sí que hem d'implementar* `toString`*.*
- *L'operació* `toString` *crida a* `getArea` *i a* `getPerimeter`*. Això és possible encara que aquestes operacions siguin virtuals pures perquè l'objecte* `p` *amb el que es cridarà* `p.toString()` **sempre serà d'una subclasse de** `RegularPolygon`**. Mai no serà directament de classe** `RegularPolygon`*. Per tant, es cridarà a les operacions* `getArea()` *i a* `getPerimeter` *de la subclasse corresponent.*

  Exemple:

  ```
  Square sq(....);
  Circumference cir(...);

  sq.toString();   //Cridara a Square::getArea()

  cir.toString();  //Cridara a Circumference::getArea()
  ```

3 La classe `Square`

1. Representa i implementa la classe `Square` (Quadrat)

   Per fer-ho, visita els **apartats 1.12, 1.13.1, 1.13.2, 1.13.3 i 1.13.4 dels apunts** i reflexiona sobre els punts següents:

   - Quins atributs i quines operacions hereta la classe `Square` de la classe `regularPolygon`?

- Quin codi tenen a la classe `Square` les operacions `getArea, getPerimeter` i `toString`?
- Has de posar a la classe `Square` les operacions `getRadius` i `getHeight`?
- La classe `Square`, a quina constructora de `RegularPolygon` crida i com la crida?

Fitxer `Square.h`:

```
class Square :public RegularPolygon{

public:
  Square();
  Square(double plength);

  MyString toString();
  double getArea();
  double getPerimeter();
};
```

Comentaris:

- *Hem d'implementar les operacions virtuals pures de* `RegularPolygon` *(*`getArea` *i* `getPerimeter`*) i redefinir* `toString`
- *En el cas d'un quadrat (square) no cal definir nous atributs als que ja hereta de* `RegularPolygon`*.*

Fitxer `Square.cc`:

```
Square::Square(){}
Square::Square(double plength)
  :RegularPolygon(plength)
{}

MyString Square::toString()
{

  return MyString("Type of polygon: Square")+
         RegularPolygon::toString();

}

double Square::getArea()
{
  return getEdgeLength()*getEdgeLength();
}


double Square::getPerimeter()
{
  return getEdgeLength()*4;
}
```

Comentaris:

- *L'operació constructora* `Square(pcolor,px,py,plength)` *es limita a cridar a la constructora de la superclasse. No ha de fer res més.*
- *L'operació* `Square::toString` *crida a* `RegularPolygon::toString` *per tal de mostrar els atributs comuns de qualsevol polígon regular (longitud aresta, àrea i perímetre).*
- *Les operacions* `getArea` *i* `getPerimeter` *adapten el càlcul d'aquests dos conceptes al cas dels quadrats.*

4 La classe Circumference

1. Representa i implementa la classe `Circumference` (Circumferència)

   Per fer-ho, visita els **apartats 1.12, 1.13.1, 1.13.2, 1.13.3 i 1.13.4 dels apunts** i reflexiona sobre els punts següents:

   - Una operació constructora d'aquesta classe haurà de rebre com a paràmetre el radi, en lloc de la longitud de l'aresta. Per què?
   - Quins atributs i quines operacions hereta la classe `Circumference` de la classe `regularPolygon`?
   - Quin codi tenen a la classe `Circumference` les operacions `getArea`, `getPerimeter` i `toString`?
   - Has de posar a la classe `Circumference` les operacions `getRadius` i `getHeight`?
   - La classe `Circumference`, a quina constructora de `RegularPolygon` crida i com la crida?

   Fitxer `Circumference.h`**:**

```
class Circumference :public RegularPolygon{

  double radius;

public:
  Circumference();
  Circumference(double pradius);

  MyString toString();
  double getArea();
  double getPerimeter();
  double getRadius();

};
```

   Comentaris:

   - *En aquest cas sí que hi ha un atribut addicional que cal incorporar a la classe* `Circumference`*:* `radius`
   - *Notem que substituim a la constructora l'atribut* `edgeLength` *per* `radius`*. En aquest cas, la longitud de l'aresta és zero (podem prendre una circumferència com un polígon regular d'infinits costats de longitud 0 cadascun). En canvi, sí que ens cal conèixer el radi de la circumferència.*
   - *Igualment apareix una operació* `getRadius`*.*

   Fitxer `Circumference.cc`**:**

```
const double PI=3.141592;

Circumference::Circumference(){radius=0;}
Circumference::Circumference(double pradius)
  :RegularPolygon(0)
{
  radius=pradius;
}

void Circumference::toString()
{
```

```
  return MyString("Type of polygon: Circumference")+
         RegularPolygon::showFeatures()+
         MyString("Radius: ")+MyString(getRadius());
}

double Circumference::getArea()
{
  return radius*radius*PI;
}

double Circumference::getPerimeter()
{
  return radius*2*PI;
}

double Circumference::getRadius()
{
  return radius;
}
```

Comentaris:

- *Notem com la crida al constructor de la classe* `RegularPolygon` *es fa amb l'inicialitzador:*
  `:RegularPolygon(pcolor,px,py,0)`
  El quart paràmetre (`edgeLength`*) és 0.*
  El codi del constructor inicialitza el radi convenientment.

5 Usant les classes

1. Ara considera la funció `f` següent **client de la jerarquia de polígons**. `f` **no és** una operació de cap classe de la jerarquia de polígons.

```
void f( (1) p)   //(1) s'ha de substituir per alguna cosa
{
 cout<<"Caracteristiques del poligon:"<<endl;
 cout<<p.toString();

 if (p.getArea()>100) cout<<"poligon gran"<<endl;
 if (p.getArea()<10) cout<<"poligon petit"<<endl;
 cout<<"----------------------"<<endl;
}
```

Considera també el programa principal següent:

```
int main()
{
 Circumference c(1,2,3,10.0); //10.0 es el radi de c
 Square s(2,4,3,8.0);         //8.0 es la longitud del costat de s

 f(c);
 f(s);
```

```
 return 0;
}
```

Prova el codi que has desenvolupat fins ara amb aquest `main` i mira si obtens el resultat següent (el format pot canviar):

```
Type of polygon: circumference
Edge length: 0.0
Radius: 10.0
Perimeter: 62.8
Area: 314.1
poligon gran
------------------------------
Type of polygon: square
Edge length: 8.0
Perimeter: 32.0
Area: 64.0
------------------------------
```

2. Per tal d'obtenir aquest resultat cal fer **dues coses molt importants**:

   (a) De quina classe cal declarar el paràmetre de `f` (1)?
   El paràmetre `p` *de* `f` *ha de declarar-se com a referència a* `RegularPolygon`*:*
   `void f(RegularPolygon& p)`
   Si es declarés com a objecte, amb un pas de paràmetres per valor:
   `void f(RegularPolygon p)`
   no funcionaria.
   També es podria declarar com apuntador:
   `void f(RegularPolygon* p)`
   Però aleshores caldria canviar una mica el codi de `f`*. Com?*

   (b) Com cal declarar les operacions `getArea, getPerimeter` i `toString`?
   Ja ho hem dit i ho hem fet: cal declarar-les com a `virtual` *a la classe* `RegularPolygon`*.*
   Si no es declaren com a `virtual` **no funcionarà** *convenientment*

6 Enriquiment de la classe `MyString`

En les seccions anteriors hem implementat les operacions `toString` suposant que disposem d'operacions constructores per convertir `int` i `double` en `MyString`.

```
MyString(int i);    //obte un MyString amb el valor d'un int
MyString(double x); //obte un MyString amb el valor d'un double
```

Exemple: `MyString(19)` genera un objecte `MyString` que modelitza la cadena de caràcters "19".

Anem a dissenyar aquestes operacions per a la classe `MyString`.

La implementació d'aquestes dues operacions pot ser la següent:

```
MyString::MyString(int i)
{
  int k,j,j2;
```

```
  char aux[8];  //hipotesi que un int t\'e menys de
                //8 digits
  bool neg=false;

  k=0; j=0;
  if (i==0) {aux[0]='0'; j=1;}
  if (i<0){neg=true;k=1; i=i*(-1);}

  while (i>0){
    aux[j]='0'+i%10;
    i=i/10;
    j++;
  }
  st=new char[j+k+1];

  if (neg) st[0]='-';

  for (j2=0;j2<j;j2++)
  {
    st[j2+k]=aux[j-j2-1];
  }
  st[k+j]='\0';
 }


MyString::MyString(double x)
{
  double fracx;
  double intx;

  st=new char[1];
  st[0]='\0';

  fracx=modf(x,&intx);

  *this= MyString(MyString((int) intx)+
                  MyString(".")+
                  MyString((int)floor(fracx*100)));
 }
```

La funció `fracx=modf(x,&intx)` *de la biblioteca* `cstring` *divideix el double* `x` *en una part entera* `intx` *i una part fraccionària* `fracx`*. Si* `x=345.455`, `fracx=455` *i* `intx=345`.