Paradigmas de la Programación
Práctico 6: Programación Orientada a Objetos (1)
conceptos fundamentales, interfaz, implementación, jerarquía, acoplamiento

Laura Alonso Alemany Ezequiel Orbe

19 de mayo de 2015

Ejercicios prácticos

1. En las siguientes declaraciones de clase, indique qué partes son implementación y qué partes son interfaz, marcando la interfaz.

```
public class URLExpSimple {

    public static void main(String[] args) {
        try {
            URL mySite = new
               URL("http://www.cs.utexas.edu/~scottm");
            URLConnection yc = mySite.openConnection();
            Scanner in = new Scanner(new
               InputStreamReader(yc.getInputStream()));
            int count = 0;
            while (in.hasNext()) {
                System.out.println(in.next());
                count++;
            }
            System.out.println("Number of tokens: " + count);
            in.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
public class Stopwatch
{
    private long startTime;
    private long stopTime;

    public static final double NANOS_PER_SEC = 1000000000.0;

        public void start()
        {       startTime = System.nanoTime();  }

        public void stop()
```

```
        {       stopTime = System.nanoTime();   }

        public double time()
        {       return (stopTime - startTime) / NANOS_PER_SEC;
           }

        public String toString(){
            return "elapsed time: " + time() + " seconds.";
        }

        public long timeInNanoseconds()
        {       return (stopTime - startTime);  }
}
```

```
public class MineSweeper
{       private int[][] myTruth;
        private boolean[][] myShow;

        public void cellPicked(int row, int col)
        {       if( inBounds(row, col) && !myShow[row][col] )
                {       myShow[row][col] = true;

                        if( myTruth[row][col] == 0)
                        {       for(int r = -1; r <= 1; r++)
                                        for(int c = -1; c <= 1;
                                           c++)
                                                cellPicked(row
                                                   + r, col +
                                                   c);
                        }
                }
        }

        public boolean inBounds(int row, int col)
        {       return 0 <= row && row < myTruth.length && 0 <=
           col && col < myTruth[0].length;
        }
}
```

```
public class PrimeEx {

public static void main(String[] args) {
        printTest(10, 4);
        printTest(2, 2);
        printTest(54161329, 4);
        printTest(1882341361, 2);
        printTest(36, 9);

        System.out.println(isPrime(54161329) + " expect false");
        System.out.println(isPrime(1882341361) + " expect
           true");
        System.out.println(isPrime(2) + " expect true");
        int numPrimes = 0;
        Stopwatch s = new Stopwatch();
```

```
        s.start();
        for(int i = 2; i < 10000000; i++) {
                if(isPrime(i)) {
                        numPrimes++;
                }
        }
        s.stop();
        System.out.println(numPrimes + " " + s);
        s.start();
        boolean[] primes = getPrimes(10000000);
        int np = 0;
        for(boolean b : primes)
                if(b)
                        np++;
        s.stop();
        System.out.println(np + " " + s);

        System.out.println(new BigInteger(1024, 10, new
           Random()));
}

public static boolean[] getPrimes(int max) {
        boolean[] result = new boolean[max + 1];
        for(int i = 2; i < result.length; i++)
                result[i] = true;
        final double LIMIT = Math.sqrt(max);
        for(int i = 2; i <= LIMIT; i++) {
                if(result[i]) {
                        // cross out all multiples;
                        int index = 2 * i;
                        while(index < result.length){
                                result[index] = false;
                                 index += i;
                        }
                }
        }
        return result;
}


public static void printTest(int num, int expectedFactors) {
        Stopwatch st = new Stopwatch();
        st.start();
        int actualFactors = numFactors(num);
        st.stop();
        System.out.println("Testing " + num + " expect " +
           expectedFactors + ", " +
                        "actual " + actualFactors);
        if(actualFactors == expectedFactors)
                System.out.println("PASSED");
        else
                System.out.println("FAILED");
        System.out.println(st.time());
}
```

```
// pre: num >= 2
public static boolean isPrime(int num) {
        assert num >= 2 : "failed precondition. num must be >=
            2. num: " + num;
        final double LIMIT = Math.sqrt(num);
        boolean isPrime = (num == 2) ? true : num % 2 != 0;
        int div = 3;
        while(div <= LIMIT && isPrime) {
                isPrime = num % div != 0;
                div += 2;
        }
        return isPrime;
}

// pre: num >= 2
public static int numFactors(int num) {
        assert num >= 2 : "failed precondition. num must be >=
            2. num: " + num;
        int result = 0;
        final double SQRT = Math.sqrt(num);
        for(int i = 1; i < SQRT; i++) {
                if(num % i == 0) {
                        result += 2;
                }
        }
        if(num % SQRT == 0)
                result++;
        return result;
}
```

2. En los siguientes programas escritos en C++, indique qué parte del programa se encontraría en el archivo .h
   El archivo .h o archivo de cabeceras contiene declaraciones de clases cuya implementación se encuentra en otros archivos, con extensión .cpp. Su finalidad es ser incluídos en otros archivos que usan las funciones que se declaran en el archivo de cabeceras. Por ejemplo, en el archivo .h tendríamos:

```
class A2DD
{
  private:
  int gx;
  int gy;

  public:
    A2DD(int x,int y); // se dejan aca las declaraciones
    int getSum();
};
```

Y en el archivo a2dd.cpp escribimos la implementación:

```
A2DD::A2DD(int x,int y) // las definiciones se prefijan con el nombre
    de la clase
{
  gx = x;
  gy = y;
}
```

```
int A2DD::getSum()
{
  return gx + gy;
}
```

```
// include header file with the class declaration
#include "frac.hpp"

// include standard header files
#include <iostream>
#include <cstdlib>

/* constructor por defecto, inicializando en 0 */
Fraction::Fraction()
 : numer(0), denom(1)
{
}

/* constructor para un numero entero, inicializando en n */
Fraction::Fraction(int n)
 : numer(n), denom(1)
{
}

/* constructor para numerador y denominador, inicializados
   segun parametro */
Fraction::Fraction(int n, int d)
 : numer(n), denom(d)
{
    // no permitimos 0 como denominador
    if (d == 0) {
        // salir del programa con un mensaje de error
        std::cerr << "error: el denominador es 0" << std::endl;
        std::exit(EXIT_FAILURE);
    }
}

/* print */
void Fraction::print()
{
    std::cout << numer << '/' << denom << std::endl;
}
```

```
#include "Circle.h" // cabecera definida por usuario en el
   mismo directorio

// Constructor
// los valores por defecto solo se especifican en la
   declaracion,
// no se repiten en la definicion
Circle::Circle(double r, string c) {
   radius = r;
   color = c;
```

```
}

// getter publico para el miembro de datos privado ``radius''
double Circle::getRadius() const {
   return radius;
}

// setter publico para el miembro de datos privado ``radius''
void Circle::setRadius(double r) {
   radius = r;
}

// getter publico para el miembro de datos privado ``color''
string Circle::getColor() const {
   return color;
}

// setter publico para el miembro de datos privado ``color''
void Circle::setColor(string c) {
   color = c;
}

// una funcion miembro publica
double Circle::getArea() const {
   return radius*radius*3.14159265;
}
```

```
#include <iostream>
#include <iomanip>
#include "Time.h"    // incluye la cabecera de la clase Time

// Constructor con valores por defecto, sin validacion de input
Time::Time(int h, int m, int s) {
   hour = h;
   minute = m;
   second = s;
}

// getter publico para el miembro de datos privado ``hour''
int Time::getHour() const {
   return hour;
}

// setter publico para el miembro de datos privado ``hour'',
   sin validacion de input
void Time::setHour(int h) {
   hour = h;
}

// getter publico para el miembro de datos privado ``minute''
int Time::getMinute() const {
   return minute;
}
```

```
// setter publico para el miembro de datos privado ``minute'',
   sin validacion de input
void Time::setMinute(int m) {
   minute = m;
}

// getter publico para el miembro de datos privado ``second''
int Time::getSecond() const {
   return second;
}

// setter publico para el miembro de datos privado ``second'',
   sin validacion de input
void Time::setSecond(int s) {
   second = s;
}

// Fijar hora, minuto y segundo, sin validacion de input.
void Time::setTime(int h, int m, int s) {
   hour = h;
   minute = m;
   second = s;
}

// Imprimir esta instancia de Time en el formato de "hh:mm:ss"
void Time::print() const {
   cout << setfill('0');    // zero-filled, need <iomanip>,
      sticky
   cout << setw(2) << hour  // set width to 2 spaces, need
      <iomanip>, non-sticky
        << ":" << setw(2) << minute
        << ":" << setw(2) << second << endl;
}

// Aumentar esta instancia por un segundo
void Time::nextSecond() {
   ++second;
   if (second >= 60) {
      second = 0;
      ++minute;
   }
   if (minute >= 60) {
      minute = 0;
      ++hour;
   }
   if (hour >= 24) {
      hour = 0;
   }
}
```

3. Dibuje la jerarquía de clases en que se basa el siguiente código.

```
class storable
{
```

```
        public:
        storable(const char*);
        virtual void read()=0; //this becomes pure virtual
           making storable an abstract
        virtual void write(); //class
        virtual ~storable();
        private:
        ....
}

class transmitter: public virtual storable
{
        public:
        void write()
        {
                read();
                ....
        }
}

class receiver: public virtual storable
{
        public:
        void read();
}

class radio: public transmitter, public receiver
{
        public:
        ...
}

int main()
{
        radio *rad = new radio();
        receiver *r1 = rad;
        transmitter *r2 =rad;

        rad->write();
        r1->write();
        r2->write();
        return 1;
}
```

```
SpinSpots spots;
SpinArm arm;

void setup() {
  size(640, 360);
  arm = new SpinArm(width/2, height/2, 0.01);
  spots = new SpinSpots(width/2, height/2, -0.02, 90.0);
}

void draw() {
```

```
  background(204);
  arm.update();
  arm.display();
  spots.update();
  spots.display();
}

class Spin {
  float x, y, speed;
  float angle = 0.0;
  Spin(float xpos, float ypos, float s) {
    x = xpos;
    y = ypos;
    speed = s;
  }
  void update() {
    angle += speed;
  }
}

class SpinArm extends Spin {
  SpinArm(float x, float y, float s) {
    super(x, y, s);
  }
  void display() {
    strokeWeight(1);
    stroke(0);
    pushMatrix();
    translate(x, y);
    angle += speed;
    rotate(angle);
    line(0, 0, 165, 0);
    popMatrix();
  }
}

class SpinSpots extends Spin {
  float dim;
  SpinSpots(float x, float y, float s, float d) {
    super(x, y, s);
    dim = d;
  }
  void display() {
    noStroke();
    pushMatrix();
    translate(x, y);
    angle += speed;
    rotate(angle);
    ellipse(-dim/2, 0, dim, dim);
    ellipse(dim/2, 0, dim, dim);
    popMatrix();
  }
}
```

```ruby
#!/usr/bin/ruby

class Being

    @@count = 0

    def initialize
        @@count += 1
        puts "Being class created"
    end

    def show_count
        "There are #{@@count} beings"
    end

end

class Human < Being

   def initialize
       super
       puts "Human is created"
   end
end

class Animal < Being

   def initialize
       super
       puts "Animal is created"
   end
end

class Dog < Animal

   def initialize
       super
       puts "Dog is created"
   end
end

Human.new
d = Dog.new
puts d.show_count
```

```
Option Strict On


Module Example

    Class Being
        Sub New()
            Console.WriteLine("Being is created")
        End Sub
```

```
    End Class

    Class Human
        Inherits Being

        Sub New()
            Console.WriteLine("Human is created")
        End Sub

    End Class

    Sub Main()

        Dim h As New Human

    End Sub

End Module
```

```
class Opt {
public:
  /// Returns true if a barrier has been hit
  virtual bool barrier(double S) =0;
  /// Returns payoff at maturity for underlying price S
  virtual double pay(double S)   =0;
};

class Vanilla:
  public Opt
{
  /// Strike
  double K;
public:
  bool barrier(double S)
  {
    return false;
  }
  double pay(double S)
  {
    return (S-K)>0 ? S-K : 0;
  }
};

class NoBarrier:
  public Opt
{
public:
  bool barrier(double S)
  {
    return false;
  }
};

class Vanilla2:
```

```
  public NoBarrier
{
  /// Strike
  double K;
public:
  double pay(double S)
  {
    return (S-K)>0 ? S-K : 0;
  }
};

class PlainPay:
  public Opt
{
  /// Strike
  double K;
public:
  double pay(double S)
  {
    return (S-K)>0 ? S-K : 0;
  }
};

class Vanilla3:
  public NoBarrier,
  public PlainPay
{

};
```

4. Dibuje la jerarquía de clases en que se basa el siguiente código.

```
class Vehicle {
public:
    explicit
    Vehicle( int topSpeed )
    : m_topSpeed( topSpeed )
    {}
    int TopSpeed() const {
       return m_topSpeed;
    }

    virtual void Save( std::ostream& ) const = 0;

private:
    int m_topSpeed;
};

class WheeledLandVehicle : public Vehicle {
public:
    WheeledLandVehicle( int topSpeed, int numberOfWheels )
    : Vehicle( topSpeed ), m_numberOfWheels( numberOfWheels )
    {}
    int NumberOfWheels() const {
```

```
        return m_numberOfWheels;
     }

     void Save( std::ostream& ) const; // is implicitly virtual

 private:
     int m_numberOfWheels;
 };

 class TrackedLandVehicle : public Vehicle {
 public:
    TrackedLandVehicle ( int topSpeed, int numberOfTracks )
    : Vehicle( topSpeed ), m_numberOfTracks ( numberOfTracks )
    {}
    int NumberOfTracks() const {
       return m_numberOfTracks;
    }
    void Save( std::ostream& ) const; // is implicitly virtual

  private:
    int m_numberOfTracks;
  };
```

```
class DrawableObject
 {
  public:
    virtual void Draw(GraphicalDrawingBoard&) const = 0; //draw
       to GraphicalDrawingBoard
 };

 class Triangle : public DrawableObject
 {
 public:
   void Draw(GraphicalDrawingBoard&) const; //draw a triangle
 };

 class Rectangle : public DrawableObject
 {
 public:
   void Draw(GraphicalDrawingBoard&) const; //draw a rectangle
 };

 class Circle : public DrawableObject
 {
 public:
   void Draw(GraphicalDrawingBoard&) const; //draw a circle
 };

 typedef std::list<DrawableObject*> DrawableList;

 DrawableList drawableList;
 GraphicalDrawingBoard drawingBoard;

 drawableList.pushback(new Triangle());
```

```
drawableList.pushback(new Rectangle());
drawableList.pushback(new Circle());

for(DrawableList::const_iterator iter = drawableList.begin(),
    endIter = drawableList.end();
    iter != endIter;
    ++iter)
{
  DrawableObject *object = *iter;
  object->Draw(drawingBoard);
}
```

5. El *Principio de sustitución de Liskov* (introducido por Barbara Liskov) es un principio de la programación orientada a objetos, y puede definirse como: Cada clase que hereda de otra puede usarse como su madre sin necesidad de conocer las diferencias entre ellas. En lenguaje mas formal: si S es un subtipo de T, entonces los objetos de tipo T en un programa pueden ser sustituidos por objetos de tipo S (es decir, los objetos de tipo S pueden sustituir objetos de tipo T), sin alterar ninguna de las propiedades deseables de ese programa (la corrección, la tarea que realiza, etc.)

   Un ejemplo que suele violar este principio es la relación de subtipado entre Cuadrado y Rectángulo, donde Cuadrado es un subtipo de rectángulo. Por qué podría suceder que los Rectángulos de un programa puedan ser sustituidos por Cuadrados y que se alteren propiedades del programa?

6. De estos dos programas, en cuál están los objetos más acoplados?

```
public class CartEntry
{
    public float Price;
    public int Quantity;
}

public class CartContents
{
    public CartEntry[] items;
}

public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        float cartTotal = 0;
        for (int i = 0; i < cart.items.Length; i++)
        {
            cartTotal += cart.items[i].Price *
               cart.items[i].Quantity;
```

```
        }
        cartTotal += cartTotal*salesTax;
        return cartTotal;
    }
}
```

```
public class CartEntry
{
    public float Price;
    public int Quantity;

    public float GetLineItemTotal()
    {
        return Price * Quantity;
    }
}

public class CartContents
{
    public CartEntry[] items;

    public float GetCartItemsTotal()
    {
        float cartTotal = 0;
        foreach (CartEntry item in items)
        {
            cartTotal += item.GetLineItemTotal();
        }
        return cartTotal;
    }
}

public class Order
{
    private CartContents cart;
    private float salesTax;

    public Order(CartContents cart, float salesTax)
    {
        this.cart = cart;
        this.salesTax = salesTax;
    }

    public float OrderTotal()
    {
        return cart.GetCartItemsTotal() * (1.0f + salesTax);
    }
}
```