FIUBA

75-08 Sistemas Operativos 2004

Código de Or. A Objetos

Ing. Osvaldo Clúa

En el mismo lugar en que encontró este texto, debió encontrar un archivo *zip* con los códigos listos para compilar y probar. Se usaron los compiladores Java se SUN (java.sun.com), C++ de la *gnu (gcc)* disponible para múltiples plataformas, entre ellas *Win* en *Cygwin* yAda95 *gnat* también disponible gratuitamente en la New York University, conviene acceder desde www.gnat.com ya que el *repository* cambia a veces de ubicación. Tambien *Cygwin* tiene una versión gratuita para Win.

Estos códigos se usarán en clase.

Ejemplo de Objetos en Java. Primero el código de operaciones sobre un Racional:

```
1.  /** Racional: Creacion y manejo de racionales */
2.
3.  class Racional{
4.     private int num=0;
5.     private int den=0;
6.     Racional(int n,int d){
7.        num=n; den=d;}
8.     int getNum() {return num;}
9.     int getDen() {return den;}
10.    void setVal(int n, int d){
11.           num=n; den=d;}
12.    private int mcd(){
13.      if (num==0) return den;
14.           else return new Racional(den%num,num).mcd();
15.      }
16.    void simplificar() {
17.      int mcd=mcd();
18.      num=num/mcd; den=den/mcd;}
19.    public String toString(){return num+"/"+den;}
20.    public Racional suma(Racional r){
21.           Racional a= new Racional(
22.                 num*r.getDen()+r.getNum()*den,den*r.getDen());
23.           a.simplificar();
24.           return a;
25.    }
26. }
```

A este programa se lo prueba con un programa del estilo de:

```
/** PruRac: Prueba de los racionales */
1.  import java.io.*;
```

```
2.  class PruRac{
3.    public static void main(String arg[]) {
4.        Racional a=new Racional(4,8);
5.        System.out.println ("El Racional a "+a);
6.        a.simplificar();
7.        System.out.println ("Simplificado  "+a);
8.        Racional b=new Racional(12,16);
9.        System.out.println ("El Racional b "+b);
10.       b.simplificar();
11.       System.out.println ("Simplificado  "+b);
12.       System.out.println(" a+b "+b.suma(a));
13.    }
14. }
```

Ahora por Herencia le agregamos los métodos para ordenarse:

```
/** Rac1: Un racional con igual, mayor y menor por herencia    */
1.  class Rac1 extends Racional{
2.    Rac1(int n, int d) {
3.      super(n,d);}
4.     boolean esIgual(Rac1 r){ return
5.         r.getNum()==getNum() && r.getDen() == getDen();}
6.     boolean esMayor(Rac1 r) { return
7.        (double) getNum() / (double) getDen() >(double) r.getNum() /(double)r.getDen();}
8.     boolean esMenor(Rac1 r) {return
9.        (double) getNum() / (double) getDen() <(double) r.getNum() /(double)r.getDen();}
10. }
```

Y su prueba:

```
/** PruRac1: Prueba de los racionales ordenables por herencia */
1.  import java.io.*;
2.  class PruRac1{
3.     public static void main(String arg[]) {
4.        Rac1 a=new Rac1(4,8);a.simplificar();
5.        Rac1 b=new Rac1(12,16); b.simplificar();
6.        System.out.println(" a>b ?="+a.esMayor(b));
7.     }
8.  }
```

En cambio puede obtenerse el mismo efecto usando una interface:

```
/* Interface Ordenable  */
1.
2.  interface Ordenable{
3.     boolean esMayor (Ordenable b);
4.     boolean esMenor (Ordenable b);
5.     boolean esIgual (Ordenable b);
6.  }

7.  /** RacOrd: Un racional con igual, mayor y menor por Interface  */
8.  class RacOrd extends Racional implements Ordenable{
9.    RacOrd(int n, int d){
10.     super(n,d);}
11.    public boolean esIgual(Ordenable r){return esIgual((RacOrd) r);}
12.    public boolean esIgual(RacOrd r){ return
13.        r.getNum()==getNum() && r.getDen() == getDen();}
14.    public boolean esMayor(Ordenable r){return esMayor((RacOrd) r);}
15.    public boolean esMayor(RacOrd r) { return
16.       (double) getNum() / (double) getDen() >(double) r.getNum() /(double)r.getDen();}
17.    public boolean esMenor(Ordenable r){return esMenor((RacOrd) r);}
18.    public boolean esMenor(RacOrd r) {return
19.       (double) getNum() / (double) getDen() <(double) r.getNum() /(double)r.getDen();}
```

```
20.         }
```

Y al hacer la prueba tenemos una ventaja de Polimorfismo:

```
            /** PruRac: Prueba de los racionales ordenables por Interface  */
1.          import java.io.*;
2.          class PruRacOrd{
3.             public static void enOrden (Ordenable a,Ordenable b){
4.                      Ordenable uno, dos;
5.                      if (a.esMayor(b)){uno=a; dos=b;}
6.                              else {uno=b;dos=a;}
7.                      System.out.println( "Se imprimen en orden: el mayor es: "+uno+
8.                              " seguido de "+dos);
9.                      }
10.
11.            public static void main(String arg[]) {
12.               RacOrd a=new RacOrd(4,8);a.simplificar();
13.               RacOrd b=new RacOrd(12,16);b.simplificar();
14.               System.out.println(" a>b ?="+a.esMayor(b));
15.                     enOrden(a,b);
16.            }
17.         }
```

El Polimorfismo esta dado en el metodo enOrden() En este caso es estático ya que se lo llama desde *main* que debe ser estático, pero esto no tiene nada que ver con el polimorfismo.

A continuación hay mas ejemplos de Polimorfismo:

En Ada95

```
--
--   Ejemplo de despacho dinamico
--
```

La especificación:

```
1.          package polis is
2.
3.          type Poli is tagged record
4.                      null;
5.             end record;
6.                procedure a(p:Poli);
7.
8.          type Uno is new Poli with
9.             record
10.                     null;
11.            end record;
12.               procedure a (u:Uno);
13.
14.
15.         type Dos is new Poli with
16.            record
17.                     null;
18.            end record;
19.               procedure a (d:Dos);
20.
21.         end polis;
```

El cuerpo

```
1.          with Ada.Text_IO, Ada.Integer_Text_IO;
2.          use Ada.Text_IO, Ada.Integer_Text_IO;
3.
4.          package body polis is
5.
6.          procedure a(p:Poli) is
```

```
7.          begin
8.             put_line("Metodo a(p) de Poli");
9.          end;
10.
11.         procedure a(u:Uno) is
12.         begin
13.            put_line("Metodo a(u) de Uno");
14.         end;
15.
16.         procedure a(d:Dos) is
17.         begin
18.            put_line("Metodo a(d) de Dos");
19.         end;
20.
21.         end polis;
22.
```

Y la prueba:

```
1.          with Ada.Text_IO, Ada.Integer_Text_IO,polis;
2.          use Ada.Text_IO, Ada.Integer_Text_IO,polis;
3.          --
4.          --   prueba de polimorfismo
5.          --
6.          procedure prueba is
7.
8.          procedure usa(p: Poli'class) is
9.          --                ~~~~~      esto implica polimorfismo
10.         begin
11.            a(p);
12.         end usa;
13.
14.         p:Poli;
15.         u:Uno;
16.         d:Dos;
17.         begin
18.         put ("usa(p:Poli) ");usa(p);
19.         put ("usa(u:Uno) ");usa(u);
20.         put ("usa(d:Dos) ");usa(d);
21.         end prueba;
```

Usando C++

```
1.          #include <iomanip>
2.          #include <iostream>
3.
4.          class Poli{
5.          public:
6.             virtual void a(){cout<<"Metodo a() de Poli"<<endl;}
7.          };
8.
9.          class Uno :public Poli {
10.         public:
11.             void a(){cout<<"Metodo a() de Uno"<<endl;}
12.         };
13.         class Dos: public Poli{
14.         public:
15.             void a(){cout<<"Metodo a() de Dos"<<endl;}
16.         };
17.         class Uso {
18.         public:     // El parametro debe pasarse por referencia
```

```
19.         void usa(Poli &x){x.a();}
20.       };
21.       void main(){
22.         Poli p;
23.         Uno u;   Dos d;    Uso s;
24.         cout<<"s.usa(p) "; s.usa(p);
25.         cout<<"s.usa(u) "; s.usa(u);
26.         cout<<"s.usa(d) "; s.usa(d);
27.       }
```

Y usando Java

```
1.        class Poli{
2.          //void poli(){};  //constructor
3.          public void a(){System.out.println ("Metodo a() de Poli");}
4.          }
5.        class Uno extends Poli{
6.          public void a(){System.out.println ("Metodo a() de Uno");}
7.          }
8.        class Dos extends Poli{
9.          public void a(){System.out.println ("Metodo a() de Dos");}
10.         }
11.       class Uso {
12.         public void usa(Poli x){x.a();};
13.         }
14.       class Prueba{
15.         public static void main(String arg[]){
16.                 Poli p=new Poli();
17.                 Uno u= new Uno();
18.                 Dos d=new Dos();
19.                 Uso s=new Uso();
20.                 System.out.print("s.usa(p) "); s.usa(p);
21.                 System.out.print("s.usa(u) "); s.usa(u);
22.                 System.out.print("s.usa(d) "); s.usa(d);
23.         }
24.       }
```

Finalmente un ejemplo de Reflection:

```
1.        import java.lang.reflect.*;
2.
3.        class Felino{
4.          int id;             // es obligatorio que los campos
5.                                  // accedidos por Reflection sean public
6.          public static boolean ladrador=false;
7.          Felino(int i){id=i;}
8.          }
9.        class Gato extends Felino{
10.         String nombre;
11.         Gato(int i,String s){super(i);nombre=s;}
12.         public String maullido(){return "Miau de "+nombre;}
13.       }
14.       class Canino{
15.         int id;
16.         String nombre;
17.         public static boolean ladrador=true;
18.         Canino (int i, String s) {id=i;nombre=s;}
19.         public String ladrido(){return "Guau de "+nombre;}
20.       }
21.       class Pekines extends Canino {
22.         Pekines(int i,String s){super(i,s);}
```

```
23.         public String ladrido(){return "Week de "+nombre+" un pekines";}
24.         public String ladrido(){return "Week de "+nombre+" un pekines";}
25.      }
26.      class GranDanes extends Canino{
27.         GranDanes(int i,String s){super(i,s);}
28.         public String ladrido(){return "WOOF de "+nombre+" un gran danes";}
29.      }
30.      class PruIndi{
31.         Object varios[];
32.      PruIndi(){
33.         varios=new Object[5];
34.         Felino tigre=new Felino(100);                    varios[0]=tigre;
35.         Gato negra=new Gato(101,"Negra");                varios[1]=negra;
36.         Canino sultan=new Canino(200,"Sultan");          varios[2]=sultan;
37.         Pekines peki=new Pekines (201,"Peki");           varios[3]=peki;
38.         GranDanes scooby=new GranDanes(202,"Scooby");varios[4]=scooby;
39.         browser();
40.         }
41.      void browser(){
42.         boolean ladra;
43.         for (int i=0;i<varios.length;i++){
44.                  Class cl=varios[i].getClass();
45.                  Field lad;
46.                  try   {
47.                           lad=cl.getField("ladrador");
48.                           ladra=lad.getBoolean(varios[i]);}
49.                  catch (Exception e){lad=null;ladra=false;}
50.                  if (ladra){System.out.println ("Objeto "+i+" ladrando:");
51.                           try{
52.                                    Method ladrar=cl.getMethod("ladrido",null);
53.                                    Object obj=ladrar.invoke(varios[i],null);
54.                                    System.out.println((String) obj);}
55.                           catch (Exception e){System.out.println("Algo raro pasa"+e);}
56.                           }
57.                  else System.out.println ("Objeto "+i+" no ladra");
58.                  }
59.         }
60.      public static void main(String a[]){
61.         PruIndi p=new PruIndi();}
62.      }
```