PRACTICO N° 5
Herencia y Polimorfismo
Duración estimada: semanas 8 y 9

Objetivos del práctico

- Implementar un diagrama de clases relacionadas por asociación, dependencia y herencia simple.
- Definir y utilizar variables polimórficas, métodos polimórficos y clases polimórficas.
- Comprender el impacto de la herencia y el polimorfismo sobre la productividad de software.
- Comprender la diferencia entre sobrecarga y redefinición, tipo estático y tipo dinámico de una variable y las restricciones sobre el polimorfismo que impone Java.
- Implementar y extender clases abstractas.

EJERCICIO 1: Implemente la clase PolizaInmuebleEscolar que hereda de PolizaInmueble, implementada en el TP N°1, de acuerdo a la siguiente especificación y considerando que los atributos de PolizaInmueble son protegidos:

PolizaInmueble
<<Atributos de instancia>> nroPoliza: entero incendio: real robo: real activa: boolean
<<Constructor>> PolizaInmueble(np: entero) PolozaInmueble(np: entero, i: real, r: real) <<Comandos>> establecerIncendio(m: real) establecerRobo(m: real) actualizarPorcentaje(p: entero) activar() desactivar() <<Consultas>> obtenerNroPoliza(): entero obtenerIncendio(): real obtenerRobo(): real obtenerCostoPoliza(): real estaActiva(): boolean

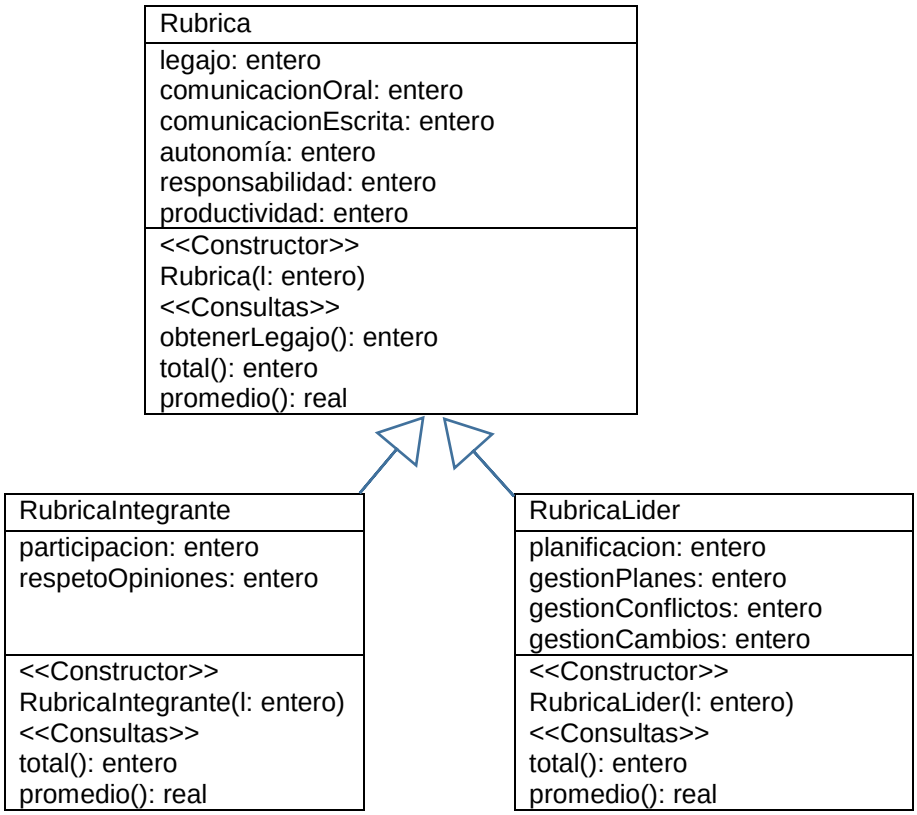
PolizaInmuebleEscolar
<<atributos de instancia>> cantAlumnos: entero cantDocentes: entero polAlumno: entero polDocente: entero
<<Constructor>> PolizaInmuebleEscolar(np: entero) PolozaInmueble(np:entero,i:real,r:real,ca,cd,a,d:entero) <<Comandos>> establecerCantAlumnos(n: entero) establecerCantDocentes(n: entero) establecerPolAlumno(n: entero) establecerPolDocente(n: entero) <<Consultas>> obtenerCantAlumnos(): entero obtenerCantDocentes(): entero obtenerPolAlumnos(): entero obtenerPolDocentes(): entero obtenerCostoPoliza(): real

- **obtenerCostoPoliza(): real** en PolizaInmuebleEscolar se computa como el costo de cualquier póliza más *polDocente* pesos por cada alumno y *polDocente* pesos por cada docente.

Implemente una clase tester para probar lo implementado.

EJERCICIO 2: Implemente una clase RefugioVip que extiende a la clase Refugio, agrega un atributo sillonesMasajeadores junto con los servicios triviales vinculados a este atributo y, además, redefine calidadRefugio() que retorna el valor fijo 100 más el número de camas. Cree una clase tester para verificar su implementación. La clase refugio es la misma que la implementada en el TP N°1.

EJERCICIO 3. Una empresa ha decidido implementar un mecanismo de evaluación por rúbrica para sus empleados. La primera versión del sistema de rúbricas ha quedado modelada por el siguiente diagrama:



a) Implemente el modelo en Java incluyendo, además, todos los métodos triviales y considerando que el constructor inicializa únicamente el legajo y todos los atributos se declaran protegidos.
b) Especificaciones:
    a. **total(): entero** Es la suma de todos los atributos de instancia. En las clases derivadas se consideran también los atributos de los ancestros.
    b. **promedio(): real** Es el promedio de todos los atributos de instancia. En las clases derivadas se consideran también los atributos de los ancestros.
c) Muestre la salida del siguiente segmento de instrucciones:

```
Rubrica r;
r = Rubrica(101);
System.out.println(r.otenerLegajo()+" "+r.total()+" "+r.promedio());
r = RubricaIntegrante(102);
System.out.println(r.otenerLegajo()+" "+r.total()+" "+r.promedio());
r = RubricaLider(103);
System.out.println(r.otenerLegajo()+" "+r.total()+" "+r.promedio());
```

EJERCICIO 4: Expliqué por qué la herencia y el polimorfismo favorecen la productividad del software.

EJERCICIO 5. Dado el siguiente segmento de código:

```
class Uno{
  int f (){
    return 0;
  }
  int g (){
   return 1;
  }
  int h () {
    return 2;
  }
}
```

```
class Tres extends Dos {
  int f (){
    return 20;
  }
  int l (){
    return 21;
  }
  int h () {
    return 12;
  }
}
```

```
class Dos extends Uno{
 int f (int x){
   return 10*x;
 }
 int g (){
   return 11;
 }
}
```

```
Uno u1, u2, u3, u4;
Dos d1, d2;
Tres t;

t = new Tres();
u1 = new Uno();
u2 = new Dos();
u3 = new Tres();
u4 = t;
d1 = new Dos();
d2 = new Tres();
```

Muestre la salida de las siguientes instrucciones:

```
System.out.println (u1.f());
System.out.println (u2.f());
System.out.println (u3.f());
System.out.println (u4.f());
System.out.println (u1.g());
System.out.println (u2.g());
System.out.println (u3.g());
System.out.println (u4.g());
System.out.println (u1.h());
System.out.println (u2.h());
System.out.println (u3.h());
System.out.println (u4.h());
```

```
System.out.println (d1.f());
System.out.println (d2.f());
System.out.println (d1.f(-1));
System.out.println (d2.f(-1));
System.out.println (d1.g());
System.out.println (d2.g());
System.out.println (d1.h());
System.out.println (d2.h());
```

```
System.out.println (t.f());
System.out.println (t.g());
System.out.println (t.h());
System.out.println (t.f(-1));
System.out.println (t.l());
```

EJERCICIO 6. Describa las restricciones sobre el polimorfismo que impone el chequeo de tipos provisto por Java.

EJERCICIO 7. Dadas las siguientes implementaciones:

```
class Alfa{
  protected int n;
  protected String s;
  public Alfa(){
    n = 10;
    s = new String ("Alfa");
  }
  public String p(int x){
    return s+" p n: "+n+" x: "+x;
  }
  public String q(int x){
    return s+" q n: "+n+" x: "+x;
  }
  public String r(int x){
    return s+" r n: "+n+" x: "+x;
  }
  public String v(char x){
    return s+" v n: "+n+" x: "+x;
  }
}
```

```
class Beta extends Alfa{
  protected int m;
  protected String ss;
  public Beta(){
    super();
    m = 11;
    ss = new String ("Beta");
  }
  public String p(int x){
    return ss+" p m: "+m+" x: "+x;
  }
  public String q(String x){
    return ss+" q m: "+m+" x: "+x;
  }
}
```

```
class Delta extends Beta{
  protected int l;
  protected char ch;
  public Delta(){
    super();
    l = 12;
    ch = 'D';
  }
  public String q(int x){
    return ch+" q l: "+l+" x: "+x;
  }
  public String t(int x){
    return ch+" t l: "+l+" x: "+x;
  }
  public String v(int x){
    String sss= ch+" v l: "+l+" x: "+x;
    return sss;
  }
}
```

Y las siguientes declaraciones de variables:

```
Alfa a1 = new Alfa();
Alfa a2 = new Beta();
Beta a3 = new Beta();
Alfa a4 = new Delta();
Beta a5 = new Delta();
Delta a6 = new Delta();
```

Indique cuáles de las siguientes instrucciones provocarán errores de compilación y de ejecución. Siempre que sea posible, muestre cuál sería la salida por consola.

```
System.out.println(a1.p(5));
System.out.println(a2.p(5));
System.out.println(a3.p(5));
System.out.println(a4.p(5));
System.out.println(a5.p(5));
System.out.println(a6.p(5));
```

```
System.out.println(a1.q(6));
System.out.println(a2.q(6));
System.out.println(a3.q(6));
System.out.println(a4.q(6));
System.out.println(a5.q(6));
System.out.println(a6.q(6));
```

```
System.out.println(a1.r(7));
System.out.println(a2.r(7));
System.out.println(a3.r(7));
System.out.println(a4.r(7));
System.out.println(a5.r(7));
System.out.println(a6.r(7));
```

```
System.out.println(a1.q("q8"));
System.out.println(a2.q("q8"));
System.out.println(a3.q("q8"));
System.out.println(a4.q("q8"));
System.out.println(a5.q("q8"));
System.out.println(a6.q("q8"));
```

```
System.out.println(a1.t(9));
System.out.println(a2.t(9));
System.out.println(a3.t(9))
System.out.println(a4.t(9));
System.out.println(a5.t(9));
System.out.println(a6.t(9));
```

```
System.out.println(a1.v(7));
System.out.println(a2.v(7));
System.out.println(a3.v(7));
System.out.println(a4.v(7));
System.out.println(a5.v(7));
System.out.println(a6.v(7));
```

EJERCICIO 8. Dadas las siguientes definiciones de clases y las declaraciones de variables

```
class Uno {
  int f (){ return 10; }
  int g (){ return 2; }
  int h () { return 100; }
  int m (int i) { return –i; }
}
```

```
class Dos extends Uno {
  int f (){ return 11; }
  int g (){ return super.g() * 2; }
  int l () { return super.f() *-1 ;}
  int n () { return f() *-5 ;}
  int m(String s) { return s.length(); }
}
```

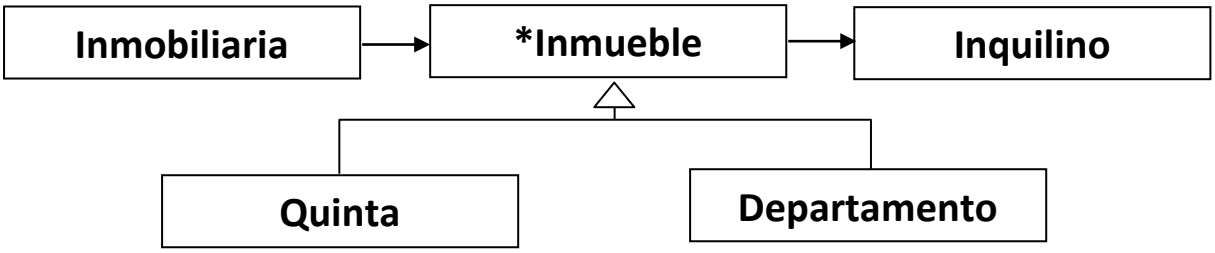
```
class Tres extends Dos {
  int f (){ return 111; }
  int h (){ return 1000 ; }
  int p() {return 10000; }
}
```

```
Uno u1, u2, u3, u4;
Dos d1, d2, d3;
Tres t1, t2;
u1 = new Uno();
u2 = new Dos();
u3 = new Tres();
t1 = new Tres();
t2 = (Tres) u3;
u4 = t1;
d1 = new Dos();
d2 = new Tres();
d3 = t2;
```

Indique qué instrucciones darán error de compilación y muestre la salida de las instrucciones válidas:

```
System.out.println("f "+u1.f()+" "+ u2.f()+" "+ u3.f()+" "+ u4.f());
System.out.println("g "+u1.g()+" "+ u2.g()+" "+ u3.g()+" "+ u4.g());
System.out.println("h "+u1.h()+"  "+ u2.h()+"  "+ u3.h()+"  "+ u4.h());
System.out.println("m  "+u1.m(8)+"  "+ u2.m(8)+"  "+ u3.m(8)+"  "+ u4.m(8));
System.out.println("f "+d1.f() +"  "+ d2.f()+"  "+ d3.f()    );
System.out.println("g "+d1.g()+"  "+ d2.g());
System.out.println(("h "+d1.h()+"  "+ d2.h());
System.out.println(("m "+d1.m(20)+"  "+ d2.m(20));
System.out.println(" f "+ t1.f()+" "+t2.f());
System.out.println(" g "+ t1.g()+" "+t2.g());
System.out.println(" h "+ t1.h()+" "+t2.h());
System.out.println(" m  "+ t1.m(3)+" "+t2.m(3));
System.out.println("l "+u2.l()+"  "+ u4.l());
System.out.println("l "+d1.l()+"  "+ d2.l());
System.out.println("n "+d1.n()+"  "+ d2.n());
```

EJERCICIO 9. Dado el siguiente diagrama de conceptos que modela las clases implementadas **en el proyecto Inmobilaria disponible en el sitio web de la materia:**



a) Agregue un método **costoAlquiler(base: entero): real**
- en la clase Inmueble el método es abstracto.
- en Departamento se calcula a partir del valor base recibido como parámetro más $1000 por metro cuadrado de superficie del inmueble, menos el 10% si el Inquilino es corporativo, más $2000 si tiene cochera.
- en Quinta se calcula a partir del valor base recibido como parámetro más $1200 por metro cuadrado de superficie del inmueble, más $500 por cada metro cuadrado de parque.

b) Agregue en las clases Inmueble y Quinta los métodos especificados a continuación:
- **precioVenta(valorMetro: entero): real** en Inmueble se calcula como la cantidad de metros cuadrados del inmueble por valorMetro.
- **precioVenta(valorMetro, valorMetroParque: entero): real** en Quinta se calcula como el precio de cualquier Inmueble más la cantidad de metros del parque por valorMetroParque.

c) Agregue en la clase Inmobiliaria los métodos:
- **costoPromedioAlquiler(base: entero): real** computa el promedio de los costos de los alquileres de todos los objetos almacenados en la colección
- **menorCostoAlquiler(m: real): Inmobiliaria** genera un objeto que contiene solo los inmbuebles cuyo costo de alquiler es menor a m.

d) Cree una clase TesterInmobiliaria para verificar los servicios definidos antes.

EJERCICIO 10. Dada la instrucción " **C v = new D();** " donde D es una clase derivada de C, indique cuáles de las siguientes afirmaciones son correctas:

a) El constructor de la clase D debe incluir una instrucción super que invoca al constructor de la clase C.
b) C es el tipo estático de la variable y determina los mensajes que el objeto ligado a la variable v puede recibir.
c) D puede redefinir cualquiera de los métodos provistos por C, excepto aquellos que se definan con el modificador "final".
d) Los métodos de la clase D puede acceder a todos los atributos y servicios definidos en C.

EJERCICIO 11. Una agencia de viajes mantiene información referida a los hoteles y cabañas que ofrece a sus clientes. La clase Alojamiento se introduce para factorizar atributos y comportamiento. En la aplicación todos los alojamientos son instancias de una de las clases derivadas: HabitaciónHotel o Cabaña (ambas clases heredan de alojamiento). La clase Agencia se implementa como una tabla. La clase cliente tiene acceso a los alojamientos por posición. Algunas posiciones pueden ser nulas.

a) Implemente el diagrama de clases.
b) Implemente una clase tester que verifique los servicios provistos por la clase Agencia.

Alojamiento*
<<atributos de instancia>> codigo: entero domicilio: String TV: boolean
<<constructores>> Alojamiento(c: entero, d: String, t: boolean) <<comandos>> establecerDomicilio(d: String) establecerTV (t: boolean) <<consultas>> obtenerCodigo(): entero obtenerDomicilio(): String tieneTV(): boolean toString (): String igualCodigo(a: Alojamiento): boolean estrellas(): entero

Agencia
<<atributos de instancia>> alojamientosAgencia: Alojamiento[]
<<constructor>> Agencia(n: entero) <<comandos>> insertarAlojamiento(a: Alojamiento, p: entero) eliminarAlojamiento(pos: entero) eliminarAlojamiento(a: Alojamiento) <<consultas>> cantAlojamientos(): entero recuperarAlojamiento(pos:entero): Alojamiento recuperarPosicion(a: Alojamiento): entero estaAlojamiento(c: entero): Alojamiento estaLlena(): boolean hayAlojamientos(): boolean masEstrellas(cantEst: entero): Agencia

HabitacionHotel
<<atributos de instancia>> precioXPersona: real piscina: boolean
<<constructores>> HabitacionHotel(c:entero,d:String,t,p:boolean, r:real) <<comandos>> establecerPrecioXPersona(r: real) establecerPiscina(p: boolean) <<consultas>> obtenerPrecioXPersona(): real tienePiscina(): boolean estrellas(): entero

Cabaña
<<atributos de instancia>> precioXDia: real cantHabitaciones: entero cantBanios: entero
<<constructores>> Cabaña(c:entero,d:String,t:boolean,p:real, ch,cb: entero) <<comandos>> establecerPrecioXDia(r: real) <<consultas>> obtenerPrecioXDia(): real

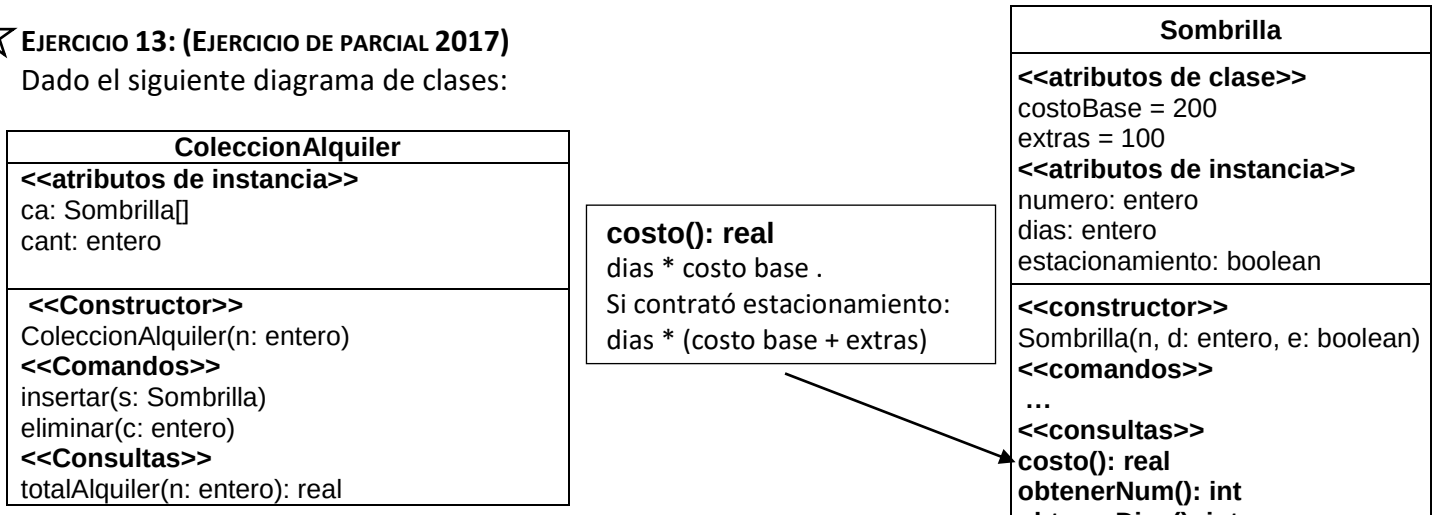
- **estrellas(): entero** en Alojamiento retorna 2 si el alojamiento tiene TV, 1 en caso contrario.
- **estrellas(): entero** en HabitacionHotel retorna la cantidad de estrellas de un Alojamiento más 1 si tiene piscina.
- **recuperarPosicion(a: Alojamiento) : entero** busca por identidad.
- **estaAlojamiento(c: entero): Alojamiento** busca por código.
- **masEstrellas(cantEst: entero): Agencia** en Agencia retorna una nueva agencia sólo con los alojamientos que tengan más de cantEst estrellas.
- **hayAlojamientos(): boolean** en Agencia retorna true si y sólo si en la tabla hay algún alojamiento.

EJERCICIO 12: Analice si las siguientes afirmaciones son correctas:

a) Una interface en Java es una clase con todos los métodos abstractos.
b) Una clase abstracta puede no tener métodos abstractos.
c) Una clase que extiende a una clase abstracta debe implementar todos los métodos abstractos heredados.
d) En método p definido en una clase A se dice sobrecargado si existe una clase B que hereda de A y brinda un método p con el mismo número y tipo de parámetros que A.
e) El ambiente de referenciamiento de una clase D que extiende a una clase C, incluye a todos los atributos de instancia declarados en D y en C.
f) Los métodos privados no pueden ser redefinidos.

☆ **EJERCICIO 13: (EJERCICIO DE PARCIAL 2017)**

Dado el siguiente diagrama de clases:



En un balneario se pueden alquilar sombrillas con capacidad para 4 personas o sombrillas especiales, más amplias, para 6 personas y servicio de camarero. En las especiales, también se pueden agregar reposeras extras abonando un costo adicional por unidad. El estacionamiento es opcional en ambos casos.

Cada nuevo alquiler se inserta al final de la colección incrementando la cantidad. **La colección NO está ordenada**. Si se almacenan *n* alquileres, se ocupan las primeras *n* componentes del arreglo.

insertar(s: Sombrilla) agrega la sombrilla pasada por parámetro a la colección. Requiere haber controlado que existe lugar disponible.

eliminar(c: entero) elimina de la colección el objeto correspondiente al número de sombrilla pasado por parámetro. La colección debe quedar compactada nuevamente.

totalAlquiler(n: entero): real devuelve el monto total que se obtendrá por las sombrillas que superen los *n* días de alquiler.

Implementar las tres clases indicadas en el diagrama.

IMPORTANTE: Los ejercicios marcados con una estrella podrán ser entregados a la cátedra para su corrección. La fecha de entrega será informada durante el cuatrimestre. Recuerden que esto es opcional y su propósito es familiarizarse con la forma que tiene la cátedra de evaluar cada tema.

Ejercicio 14: Identifique errores de compilación, ejecución y aplicación en el siguiente código. A medida que vaya detectando cada error elimine la instrucción o realice una modificación de modo pueda continuar con el análisis de lo que sigue.

```
abstract class Alfa{
        protected int i;
        protected boolean b;
        public Alfa () {
                i = 10;
                b= true;
        }
        abstract public int p(int j) ;
        public int q(int j) {
                return  i+j;
        }
        public int r(int j) {
                //retorna la suma de todos los
        números entre i y j
                int s = 0;
                int m = 0;
                for (int m = i; m <= j; m++);
                        s = s + m;
                return  s;
        }
 }
```

```
class Delta extends Alfa {
        private double k;
        public Delta () {
                super();
                k = 1.5;
        }
        public int p(int x){
                return (int)
                k * i + x;
        }
        public int q(int x,int y) {
                int z = (int) (x * k);
                return  z+y;
        }
}
```

```
class Beta extends Alfa {
        protected int k;
        public Beta () {
                k = 2;
                super();
        }
        public int P(int j){
                return j * k;
        }
        public int Q(int j) {
                return  i * j * k;
        }
 }
```

```
class Gama extends Delta {
        protected int a;
        public Gama () {
                super();
                a = -1;
        }
        public double s(int x){
                return  k * a * x;
        }
}
```

```
class Aplicacion {
        public static void main (String a []){
                Gama g1 = null;
                Gama g2 = null;
                Gama g3 = null;
                Delta d = null;
                Beta b = null;
                Alfa a;
```

```
                Alfa e1 = null;
                Alfa e3 = null;
                Alfa e2 = e3;
                int  i;

                e2= new Alfa();
                i = e2.q(3);

                e3 = new  Delta ();
                i = e3.q(3);
                i= e3.q(4,5);

                Alfa b = new Delta();
                i = b.q(3);
                b = d;

                d = new Delta();
                i = d.q(4,5);
                i = d.s(2);
                d = g1;

                double z;
                g1 = new Delta();
                z = g1.s(2);
                g2 = new Gama();
                z = g2.s(2);
                g1 = g3;
                g1.p(2);
        }
}
```