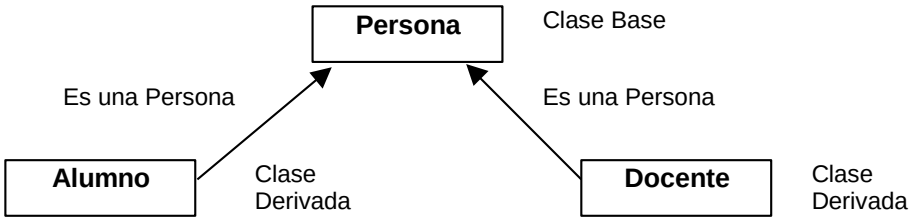
UNIDAD III: HERENCIA y POLIMORFISMO

En esta notas de clase, se presenta el concepto de herencia y su utilización desde el lenguaje de programación Java. Por último, se presentará el concepto de polimorfismo y su relación con la herencia. Los temas que se tratados son:

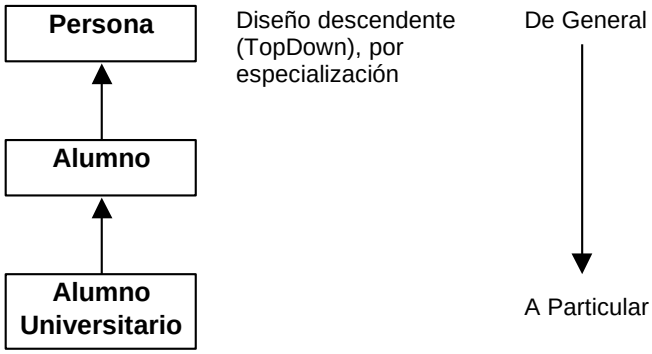
* Herencia de clases.
* Sobrecarga.
* Polimorfismo.

En la programación orientada a objetos, la **herencia** permite que una clase herede las características y el comportamiento de otra clase (sus métodos y atributos) y a continuación, modificar este comportamiento según sean las necesidades.

Esto permite tener varias clases con un comportamiento similar en ciertas situaciones (por ejemplo, al llamar a ciertos métodos) y un comportamiento específico en otras. Como se muestra en la figura.



Las distintas clases de un programa se organizan mediante la ***jerarquía de clases.***



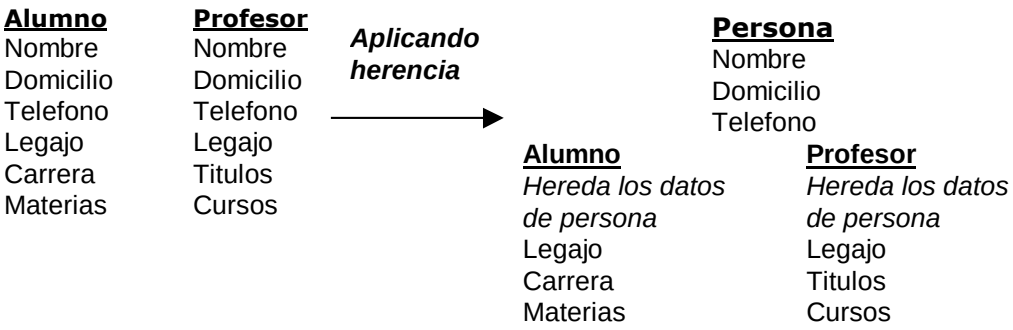
Cómo aplicamos la herencia, en el lenguaje Java???

Para heredar de una clase se ha de incluir la palabra reservada **extends** seguida de la clase de la que se quiere heredar. *En Java solamente se puede heredar desde una clase*, al contrario que en otros lenguajes de programación, como C++, donde existe la herencia múltiple, en la que se puede heredar de varias clases.

La forma que suele presentar una clase que hereda de otra es la siguiente:

Modo Acceso *class* NombreClase ***extends*** ClasePadre ...

En el esquema que se muestra a continuación, se plantean dos entidades una Alumno y otra Profesor. Si aplicamos el concepto de Herencia, surgen otras tres entidades:



La implementación en Java sería:

Ejemplo:

```
public class Persona  → Clase Base o Madre
{
        //Atributos
        //Métodos  }
 //Declaración de las clases Hijas o Derivadas de Persona.

 public class Alumno extends Persona
{       //Atributos
        //Métodos
}


public class Profesor extends Persona
{       //Atributos
        //Métodos
}
```

Cuando una clase hereda de otra, la clase hija hereda los atributos y los métodos de la clase padre, pero existen ciertas restricciones en esta herencia:

a) Los atributos y los métodos con modo de acceso **private** no se heredan.

b) Se heredan los atributos y los métodos con modo de acceso public y protected.

c) No se hereda un atributo de una clase padre si en la clase hija se define un atributo con el mismo nombre que el atributo de la clase padre.

d) No se hereda un método si éste es sobrecargado.

e) No se heredan los **contructores** de la clase base o madre. Por lo tanto en el constructor de la clase derivada, en la primera línea del mismo, hay que invocar al constructor de la clase base con la palabra **super** y la cantidad de parametros que requiera el constructor, esto es así porque primero se debe crear la clase base y por último la clase derivada.

Para referenciar a los miembros de la clase base se utiliza la palabra reservada **super**, generalmente ésta se usa si los miembros de la clase base se llaman de la misma forma que los de la clase derivada, sino, no hace falta usar la palabra **super**.

Sobrecarga y ocultación

Al crear una clase que hereda de otra se pueden definir nuevos atributos y métodos o se puede modificar el comportamiento de los objetos de la clase sobrecargando los métodos de la clase padre. La sobrecarga se refiere a la posibilidad de proporcionar nuevos métodos con el mismo nombre que tienen los métodos de la clase padre, pero con diferente comportamiento. Esto permite que el entorno y comportamiento de una clase sea heredado por otra y solamente se modifiquen aquellos métodos que sean necesarios para su comportamiento especifico.

El operador instanceof

Se puede consultar si un objeto pertenece a una clase mediante:

```
Object obj;
...
if (obj instanceof A)
  // obj pertenece a la clase A
  A a=(A)obj;  // Ok, nunca hay error
```

Los objetos de la clase B también son instancias de la clase A:

```
Object obj= new B();
if (obj instanceof A) // true
  A a= (A)obj;       // Ok
```

POLIMORFISMO

El polimorfismo indica que una variable puede adoptar múltiples formas. Cuando se habla de polimorfismo en programación orientada a objetos se suelen entender dos cosas:

La primera suele referirse a que se puede trabajar con un objeto de una clase sin conocer ni importar de qué clase se trata. Es decir, se trabajará igual sea cual sea la clase a la que pertenece el objeto. Esto normalmente se consigue mediante jerarquías de clases o interfaces.

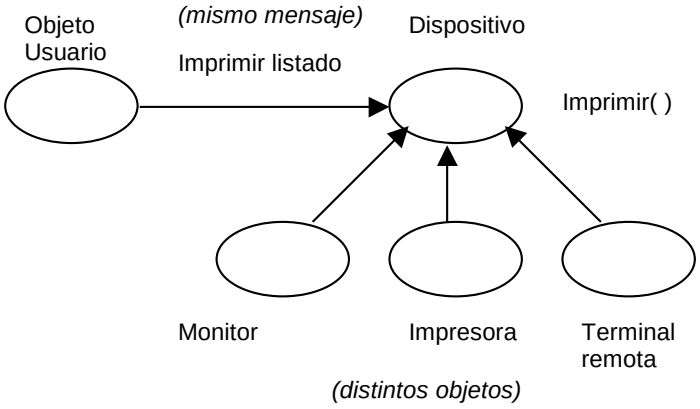
La segunda se suele referir a la posibilidad de declarar métodos con el mismo nombre que pueden tener diferentes argumentos dentro de una misma clase.

Un ejemplo de la primera podría ser la utilización de un método como toStringO, que tienen todos los objetos al heredar de la clase Object.

Resumiendo: La facultad de llamar a una variedad de métodos utilizando el mismo medio de acceso, proporcionados por los métodos redefinidos en las clases hijas, es denomida polimorfismo.

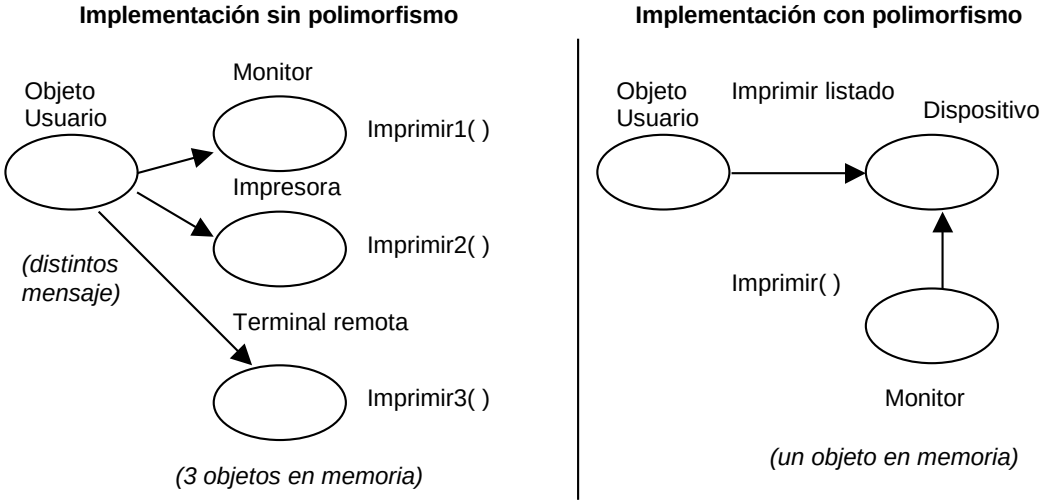
La palabra "polimorfismo" significa "la facultad de asumir muchas formas" refiriéndose a la facultad de llamar a muchos métodos diferentes utilizando una única sentencia.

Cuando se invoca a una método que esta definido en la clase madre o base y redefinido en las clases derivadas o hijas, la versión que se ejecuta del método depende de la clase del objeto referenciado, no de la variable que lo referencia.



Permite mejorar la implementación de los programas. Por ejemplo, la implementación del ejemplo anterior sin usar polimorfismo, implicaría crear la cantidad de objetos que podría necesitar el usuario, si bien el mismo solo va a usar uno por vez, tiene que tenerlos a todos en la memoria para que cuando requiera el listado correspondiente, tenga acceso a cualquier salida. En cambio si se utiliza polimorfismo, en memoria sólo se requerirá en forma permanente una referencia al dispositivo (genérico) y cuando el usuario decida imprimir el listado, elegirá en que dispositivo lo hará (monitor, impresora o terminal remota), por lo tanto en tiempo de ejecución se decide cual es el objeto que

estará activo en el momento de la impresión, dejando en memoria solo el objeto que se requiera y no los n posibles.



Polimorfismo en Java

En Java, hay ciertos conceptos que se van a usar para aplicar polimorfismo, ellos son:

Clases abstractas : ***Son clases genéricas, sirven para agrupar clases del mismo género, no se pueden instanciar, es decir no se pueden crear objetos a partir de ellas, ya que representan conceptos tan generales que no se puede definir como será la implementación de la misma. Es por eso que dichas clases llevan métodos que se llaman abstractos, que no tienen implementación.***

Sintaxis:

Acceso **abstract** class NombreClase
{...}

Por ejemplo:

public **abstract** class Dispositivo
{ ..... }

En Java una clase abstracta debe tener al menos un método abstracto.

Métodos abstractos : ***Métodos que no tienen implementación, se utilizan para recibir los mensajes en común. No tiene implementación porque en el nivel donde se definen no hay información suficiente para implementarlos. En los próximos niveles de la jerarquía se pueden implementar, para ello deben ser redefinidos en niveles posteriores.***

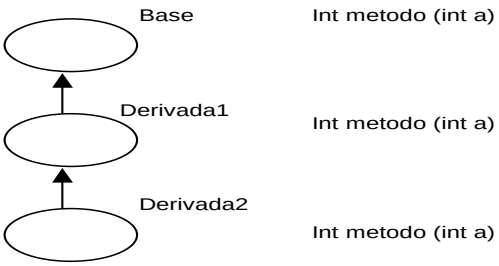
Sintaxis:

Acceso **abstract** tipo NombreMetodo (parametros);

Por ejemplo:

```
public abstract void Imprimir();
```

Métodos redefinidos : ***Métodos que tienen la misma firma (tipo, nombre y argumentos) en los distintos niveles de una jerarquía.***



Implementación en Java

1. Diseñar e implementar las clases: Diseñar una jerarquía de clases con las clases que intervendrán en el polimorfismo, dicha jerarquía debe tener una clase abstracta como clase base y tantas clases derivadas de ella, como haga falta.
2. Elegir él o los métodos que recibirán el mismo mensaje, definirlo en la clase base como abstracto y redefinirlo en las clases derivadas con la implementación correspondiente.
3. En la clase usuaria, puede ser la que lleve el método main(), realizar lo siguiente:
    1. Definir una variable de la clase base abstracta
    2. Solicitarle al usuario que elija que objeto va a crear
    3. Crear el objeto de acuerdo a la opción del usuario, la referencia de dicho objeto se guarda en la variable crada en primera instancia.

Por ejemplo:

Usando un solo objeto polimórfico

```
Dispositivo d;
char c;
 // elija el dispositivo
c = In.readChar();
if (c == 'M')
   d = new Monitor(...);
else
   if (c == 'I')
     d = new Impresora(...);
   else
     d = new Terminal(...);

d.Imprimir();
```

Ahí se aplica el polimorfismo, el objeto que esté creado recibirá el mensaje y lo ejecutará. En tiempo de programación no importa si es un monitor, impresora o terminal. En tiempo de ejecución se decidirá cual es el objeto que se utilizará

Usando varios objetos polimórfico (vector)

```
Dispositivo [ ] d;
//crea el vector de referencias
d = new Dispositivo[10];
char c;   int i;
//crear los objetos
for (i = 0;  i < 10; i++)
{
    // elija el dispositivo
    c = In.readChar();
    if (c == 'M')
      d[i] = new Monitor(...);
   else
      if (c == 'I')
        d[i] = new Impresora(...);
      else
        d[i] = new Terminal(...);
}
//usar los objetos
for (i = 0;  i < 10; i++)
   d[i].Imprimir();
```

PARTE PRACTICA

HERENCIA

1. El departamento de recursos humanos, necesita procesar información de los profesionales que trabajan en la empresa.
En la etapa de relevamiento, se han detectado las siguientes entidades:

Persona: dni, nombre, dirección, tipo de documento.(0-DNI/1-LE/2-LC/3-otro)
Profesional: profesión(0-abogado/1-Ingeniero/2-contador/3-Otro), puesto, sueldo, matrícula.

Almacenar la información en un arreglo, definir en la clase contenedora constructor por defecto y argumento.
Se pide:

Informar la cantidad de personal, discrimado por profesión.(4 totales)
Mostrar el nombre del contador que tiene menor sueldo.
Informar el importe total de sueldos abonado por la empresa al personal.

Resolución.

```
//Persona.java
public class Persona
{  private long dni;
   private String nom;
   private String dir;
   private int tipo;

  public Persona()
  {  dni=0;
     nom="";
     tipo=0;
    dir="";
   }

   public Persona(long d,String n, String di, int t)
  { dni=d;
     nom=n;
     dir=di;
    tipo=t;
   }

 public long getDni()
 { return dni; }

 public void setDniCod(long x)
 { dni=x; }

  public String getNom()
 { return nom; }

 public void setNom(String n)
 { nom=n; }

 public String getDir()
```

```
 { return dir; }

 public void setDir(String x)

 {  dir=x; }

 public void setTipo (int x)
 {tipo=x;}
 public int getTipo()
 { return tipo; }

 public String toString()
 {  return "\nDni: "+dni+"\nTipo de Dni: "+tipo+"\nNombre: "+nom+"\nDireccion:  "+dir;

 }
}//Fin de la Clase Persona


//Profesional.java
public class Profesional extends Persona
{      private int prof;
       private int matr;
       private float sueldo;
       private String puesto;

  public Profesional ()
 { super(); //invoco al constructor por defecto de Persona
    puesto="";
    prof=0;
    matr=0;
    sueldo=0;
    }

 public Profesional (long d,String n, String di, int t, int pr, int m,float s, String p)
 { super(d,n,di,t); //invoco al constructor con argumento de Persona
    prof=pr;
    matr=m;
    sueldo=s;
    puesto=p;
   }

 public void setProf (int x)
 {prof=x;}
 public void setSueldo (float x)
 {sueldo=x;}
 public void setMatr (int x)
 {matr=x;}
 public void setPuesto (String x)
 {puesto=x;}
 public int getProf ()
 {return prof;}
 public int getMatr ()
 {return matr;}
 public String getPuesto ()
 {return puesto;}
 public float getSueldo ()
 {return sueldo;}

 public String toString()
 {   return   super.toString()   +   "\nProfesion:"+prof+"\nNro   Matricula:   "+matr+"\nSueldo:
"+sueldo+"\nPuesto: "+puesto; }
}
//fin clase Profesional


//Clase Contenedora Lista
//ListaPersonal.java
```

```
public class ListaPersonal
{ private Persona p[];

  public ListaPersonal ()
  {p=new Persona [50];}

  public ListaPersonal(int cant)
  { p=new Persona [cant]; }


 public void setPersona(int indice,Persona x)
  { p[indice]=x; }

  public Persona getPersona (int indice)
  { return p[indice];}

  public String CantidadDeProfesionales()
  { int c1=0,c2=0,c3=0,c0=0;

     for(int i=0; i<p.length;i++)
       {if (p[i]!= null && (p[i] instanceof Profesional ))
       {
         Profesional pr  = (Profesional) p[i];
         switch (pr.getProf())
         {
         case 0:c0++; break;
         case 1:c1++;break;
         case 2:c2++;break;
         case 3:c3++;break;
         }
       }
       }
     return "\nAbogados: "+c0+"\nIngeniero: "+c1+"\nContador: "+c2+"\nOtro: "+c3;

  }
  public String ContadorMenorSueldo ()
  {    int b=0;
             float menorSueldo=0;
             String menorNombre="\nNo hay contadores en la empresa";
     for(int i=0; i<p.length;i++)
             {if (p[i]!= null && (p[i] instanceof Profesional ))
                    {
                    Profesional pr  = (Profesional) p[i];
                    if (pr.getProf()==2)
                           {if (b==0||pr.getSueldo()<menorSueldo)
                                  {

                                         menorSueldo=pr.getSueldo();
                                         menorNombre="El nombre del empleado de menor sueldo es :
"+pr.getNom();
                                         b=1;
                                  }
                           }
                    }
       }
     return menorNombre;

  }
   public float importeTotal ()
  { float total=0;
     for(int i=0; i<p.length;i++)
       {if (p[i]!= null && (p[i] instanceof Profesional ))
       {
         Profesional pr  = (Profesional) p[i];
         total+=pr.getSueldo();

       }
```

```
       }
     return total;

  }
}
//fin Clase ListaPersonal

//Aplicacion.java
public class Aplicacion
{ public static void main(String args[])
  { int tipo,c,prof,mat;
      long dni;
     float sueldo;
     String nom,dire,puesto;

     ListaPersonal lp;
     Persona p;

     System.out.println("Ingrese cantidad de Empleados:");
     c=In.readInt();
     lp=new ListaPersonal(c);

     for(int i=0;i<c;i++)
     {  System.out.println("\nIngrese num de documento:");
        dni=In.readLong ();
        do
        {
        System.out.println("Tipo de documento: (0-DNI 1-LE 2-LC 3-Otro) ");
       tipo=In.readInt();
       }
        while (tipo<0||tipo>3);
        System.out.println("Nombre:");
        nom=In.readLine ();
        System.out.println("Direccion:");
        dire=In.readLine();
             do
        {
        System.out.println("Profesion: (0-Abogado 1-Ingeniero 2-Contador 3-Otro) ");
       prof=In.readInt();
       }
        while (prof<0||prof>3);
        System.out.println("Puesto:");
        puesto=In.readLine();
             System.out.println("Matricula:");
        mat=In.readInt();
             System.out.println("Sueldo:");
             sueldo=In.readFloat();

              p=new Profesional (dni,nom,dire,tipo,prof,mat,sueldo,puesto);
         lp.setPersona(i,p);
    }
        System.out.println("\n\tRESULTODOS\n\t***********");
        System.out.println("\nCantidad de Profesionales: "+lp.CantidadDeProfesionales());
        System.out.println(lp.ContadorMenorSueldo());
        System.out.println("\nEl Total de sueldo pagados por la empresa es:"+lp.importeTotal());
        }

}//Fin Aplicacion
```

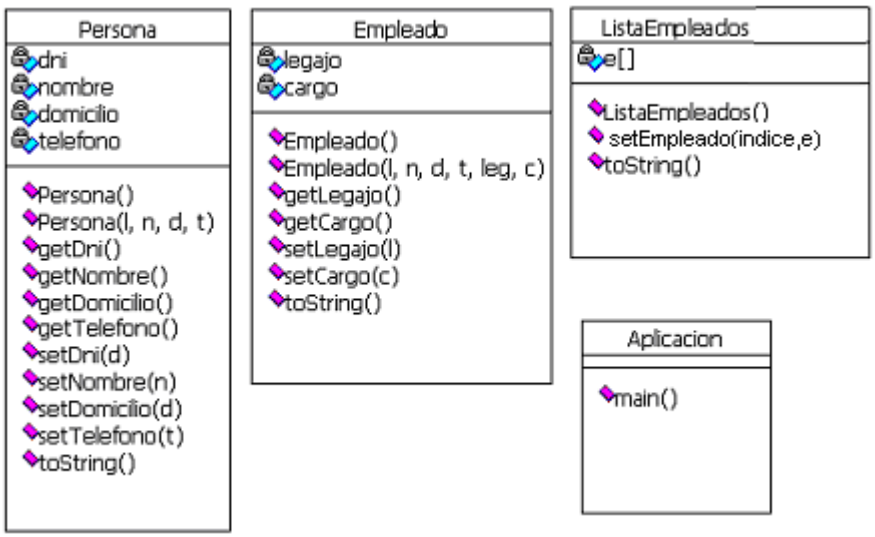
2. Se necesita procesar información de los datos empleados de una empresa. En la etapa de relevamiento se detectan las siguientes entidades:

Persona: dni, nombre, domicilio, teléfono.

Empleado: es una persona, y tiene los siguientes atributos: legajo, cargo.

Almacenar los datos de los empleados en un arreglo, e informar todos los datos de cada uno de ellos. La cantidad de empleados que tiene la empresa son 30.

Diagrama de clases



Implementación en java

```
//Empleado.java
public class Persona
{
	private long dni;
	private String nombre;
	private String domicilio;
	private String telefono;

	public Persona()
	{
		dni = 0;
		nombre = "";
		domicilio="";
		telefono = "";
	}

	public Persona(long l, String n, String d ,String t)
	{
		dni = l;
		nombre = n;
		domicilio=d;
		telefono = t;
	}

	public void setDni(long d)
	{ dni = d; }
```

```
	public void setNombre(String n)
	{ nombre = n; }

	public void setDomicilio(String d)
	{ domicilio = d; }

	public void setTelefono(String t)
	{ telefono = t; }

	public long getDni()
	{ return dni; }

	public String getNombre()
	{ return nombre; }

	public String getTelefono()
	{ return telefono; }

	public String getDomicilio()
	{ return domicilio; }


	public String toString()
	{
		String aux=" ";
		aux = dni+ ", "+nombre+", "+ domicilio+", "+telefono;
		return aux;
	}
}//fin de la clase

//Empleado.java
class Empleado extends Persona
{
	private int legajo;
	private String cargo;

	public Empleado()
	{
		super ();    //llama al constructor por defecto de persona
		legajo = 0;
		cargo = " ";
	}

	public Empleado(long l, String n, String d,String t, int leg, String c)
	{
		super (l,n, d, t);  //llama al constructor con parámetros de persona
		legajo = leg;
		cargo = c;
	}

	public void setLegajo(int l)
	{ legajo = l; }

	public void setCargo(String c)
	{ cargo = c; }

	public int getLegajo()
	{ return legajo; }

	public String getCargo()
	{ return cargo; }

	public String toString()
	{
```

```
            String aux;
            aux = legajo+ ", ";

            aux += super.toString();
            aux +=", "+cargo;

            return aux;
      }
}//fin de la clase

//ListaEmpleados
public class ListaEmpleados
{ private Empleado e[];

  public ListaEmpleados()
  { e=new Empleado[30];
    for(int i=0;i<30;i++)
       e[i]=null;
  }

  public void setEmpleado(int indice,Empleado emp)
  { e[indice]=emp; }

  public Empleado getEmpleado(int indice)
  { return e[indice]; }

  public String toString()
  { String aux= "";
    for(int i=0;i<e.length;i++)
        aux+=e[i].toString()+", ";
    return aux;
  }
}//Fin de la Clase ListaEmpleados


//Aplicación.java
public class Aplicacion
{     public static void main (String args[])
      {     int c,l,d;
            String n,dir,t,car;

            //declara dos referencias a objetos alumno
            ListaEmpleados le;
            Empleado e;

            System.out.print("Ingrese la cant. empleados de la empresa:");
            c=In.readInt();
            //crea el objeto
            le = new ListaEmpleados(c);

            //Hago la carga del arreglo.
            for(int i=0;i<c;i++)
            { //carga valores para el empleado.
              System.out.print("Ingrese el DNI del empleado:");
              d = In.readInt();
              System.out.print("Ingrese el nombre: ");
              n = In.readString();
              System.out.print("Ingrese dirección: ");
              dir = In.readString();
              System.out.print("Ingrese el telefono: ");
              t = In.readString();
              System.out.print("Ingrese el legajo : ");
              l = In.readInt();
              System.out.println("Ingrese cargo:");
              car=In.readString();

               //crea el objeto Empleado
```

```
                e = new Empleado(d,n,dir, t, l,car);
                le.setEmpleado(i,e);
              }

            //Visualizacion de los datos de cada empleado.
             System.out.println("Los datos del empleado son:"+le.toString());}

}//fin clase aplicacion
```

3. En un sistema de comercialización de automóviles, se encontraron las siguientes entidades durante la etapa de análisis:

Vehículo: entidad genérica que posee los siguientes atributos: denominación, marca, stock, precio.
Automóvil: es un Vehículo, tiene como atributo cant. puertas.
Almacenar los datos de los vehículos en un arreglo. Se desconoce la cantidad total de vehículos de la empresa.
Se pide:

Informar la denominación del automóvil que tiene un mayor precio.
Informar el total de stock de automóviles de la empresa.
Informar el nombre de los automóviles, cuyo stock supera un valor "x". El mismo deberá ser ingresado por teclado.

4. Una empresa constructora de viviendas, necesita procesar información de las casas que construye.
Se detecta la siguiente organización de los datos:

Vivienda: código de vivienda, domicilio, valuación superficie edificada, base imponible.
Casa: superficie del terreno.

Almacenar la información en un arreglo, definir en la clase contenedora constructor por defecto y argumento.
Se pide:
Informar el código de vivienda y domicilio que tiene mayor superficie edificada.
Desarrollar un método calcular_precio(cod_vivienda), que permita calcular el precio que se obtiene de multiplicar la base imponible por la valuación dividido por la superficie edificada.

PARTE PRACTICA

POLIMORFISMO

1. Un comercio necesita conocer lo recaudado de las ventas que realiza. Para ello se han detectado las siguientes entidades.

Venta: nro comprobante, fecha, cliente, importe bruto. Esta entidad deberá incluir un método que retorna MontoTotalVta, que será calculado según el tipo de venta. No se harán instancias de esta clase.

Venta Efectivo: Es una venta y agrega como dato un porcentaje. Este porcentaje se aplica disminuyendo el importe total de la venta en efectivo.

Venta Tarjeta Crédito: Es una venta y agrega como dato tipo. Este dato representa con que tarjeta realiza la transacción (1-Visa 2-Naranja 3-Otra). En el caso de operar con la tarjeta 1, el importe de la venta tiene un recargo del 1,5% si es 2 de 2,5% y si es 3 tiene un recargo del 5%.

Se pide:

a. Implementar las clases enunciadas, aplicando los conceptos de herencia y polimorfismo.

b. Crear en el main(), un arreglo de objetos que almacene las ventas efectuadas por el comercio. Y mostrar de cada venta su monto total de la venta, en función si es contado o con tarjeta de crédito. Además calcular cuantas ventas son al contado a con tarjeta.

```
//Resolución
public abstract class Venta
{ private int nro;
private String fecha;
private String cliente;
private float importe;
public Venta(int n,String f, String c,float i)
{ nro=n;
fecha=f;
cliente=c;
importe=i;}
public float getImporte()
{ return importe;}
public void setImporte(float i)
{ importe=i;}
//Demás met. get() y set()
public abstract float CalcularImporteTotal(); //Método abstracto.
public String toString()
{ return nro+" "+fecha+ "" +cliente+ ""+importe;}
}

public class VentaEfectivo extends Venta
{ private int porcentaje;
public VentaEfectivo(int n,String f,String c,float i, int p)
{ super(n,f,c,i); //Invoco al constructor clase base.
porcentaje=p;
}
public int getPorcentaje()
{ return porcentaje;}
public void setPorcentaje(int porc)
{ porcentaje=porc;}
//Implemento el método abstracto.
public float CalcularImporteTotal ()
{ return getImporte()-(getImporte()*porcentaje/100);}
public String toString()
{ return (*2) toString()+ ""+CalcularImporteTotal();}
}


//*2. Los métodos que son públicos puede omitir la palabra super. También puede
//colocar super.toString().
```

```
public class VentaTarjetaCredito extends Venta
{ private int tipo;
public VentaTarjetaCredito(int n,String f,String c,float i,int t)
{ super(n,f,c,i); //Invoco al constructor clase base.
tipo=t;
}
public int getTipo()
{ return tipo;}
public void setTipo(int t)
{ tipo=t;}
//Redefinicion del metodo abstracto corresp. a la clase base.
public float CalcularImporteTotal()
{ float p=getImporte() ;
switch(tipo)
{ case 1: p+= (p * 1.5 / 100);
break;
case 2: p+= (p * 2.5 / 100);
break;
case 3: p+= (p * 5 / 100);
break;
}
return p;
}
public String toString()
{ return toString()+ ""+CalcularImporteTotal();}
} //Fin de la Clase VentaTarjetaCredito

//Aplicación.java
public class Aplicacion
{ //Creo 3 objetos 1 Venta Contado y otro Venta Tarjeta Credito.
public static void main(String args[])
{ //Declaración de variables.
int nro,porc,tip,cont,opcion;
String fec,cli;
float imp;
VentaEfectivo ve=null;
VentaTarjetaCredito vt=null;
cont=0;
//No uso un clase contenedora y hago un arreglo de 4 ventas, que puede ser
// Efectivo o Tarjeta Credito.
Venta ven[];
ven=new Venta[4];
//Cargo el arreglo que puede contener VentaContado o TarjetaCredito.
for(int i=0;i<4;i++)
{ //Ingreso datos comunes a las Ventas.
System.out.println("Ing. Nro comprobante");
nro=In.readInt();
System.out.println("Ing. Fecha:");
fec=In.readString();
System.out.println("Ing. Cliente:");
cli=In.readString();
System.out.println("Ing. importe:");
imp=In.readFloat();
System.out.println("Venta desea cargar?(1-Contado 2-Tarjeta):");
opcion=In.readInt();
if (opcion==1)
{ System.out.println("Ing. porcentaje:");
porc=In.readInt();
ve=new VentaEfectivo(nro,fec,cli,imp,porc);
ven[i]=ve; /*1*/
}
else
{ System.out.println("Ing. Tipo tarjeta(1,2,3):");
tip=In.readInt();
vt=new VentaTarjetaCredito(nro,fec,cli,imp,tip);
ven[i]=vt;/*1*/}
```

```
}//Fin del For.
//Ahora Aplicamos el Polimorfismo.Para cada venta si es contado o tarjeta
//mostramos el monto total.
//Además mostramos cuantas ventas fueron con Tarjeta.
for(int i=0;i<4;i++)
{System.out.println("El monto de la venta es:" + ven[i].
CalcularImporteTotal());
if (ven[i] instanceof VentaEfectivo)
cont++;
}
System.out.println("La cant. de venta contado es:"+cont);
}//Fin del main()
}//Fin de la clase Aplicación.
```

```
*1 Este ejemplo se define una arreglo unidimesnional del tipo Venta con 4 elementos, que el
lenguaje Java lo inicializa en null.
Despues se creará un objetos de clases hija y se almacena su referencia en el elemento que
corresponda. En este momento Java realiza un conversión implicita del tipo de referencia devuelta
por el operador new al tipo Venta.
```

2. Una fábrica de dulces desea implementar un Sistema de Producción donde se manejan los siguientes datos:

Golosina: código de golosina, nombre, precio, ingredientes (String)

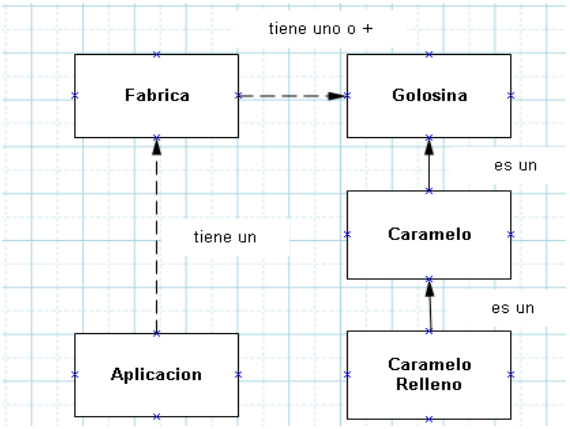
Caramelo: entidad que agrega datos de golosina y posee los siguientes atributos: sabor, color.

Caramelo Relleno: agrega datos de caramelo y posee los siguientes datos: sabor relleno, cantidad.

Además existe un entidad que agrega **Fabrica** de Golosinas, que posee como atributo un arreglo de tipo Golosina(Caramelo o Caramelo Relleno)

Se pide:

Realizar el Diagrama de Clases.



Desarrollar un método que permita mostrar los sabores de los caramelos rellenos.

Resolución.

```
//Implementación en Java
//Golosina.java
public class Golosina
{ protected int cod;
   protected String nom;
   protected float pre;
```

```
   protected String ingred;

  public Golosina()
  { cod=0;
     nom="";
     pre=0;
    ingred="";
   }

   public Golosina(int c,String n, float p, String i)
  { cod=c;
     nom=n;
     pre=p;
    ingred=i;
   }

 public int getCod()
 { return cod; }

 public void setCod(int c)
 { cod=c; }

  public String getNom()
 { return nom; }

 public void setNom(String n)
 { nom=n; }

 public float getPre()
 { return pre; }

 public void setPre(float p)
 {  pre=p; }

 public String getIngred()
 { return ingred; }

 public String toString()
 { String aux=" ";
   aux=cod+ " "+nom+" "+pre+ " "+ingred;
   return aux;
 }
}//Fin de la Clase Golosina
```

//Caramelo.java

```
public class Caramelo extends Golosina
{ private String sabor;
   private int cant;

 public Caramelo()
 { super(); //invoco al constructor por defecto de golosina
    sabor="";
    cant=0; }

 public Caramelo(int c,String n, float p, String i ,String s,int ca)
 { super(c,n,p,i); //invoco al constructor con argumento de golosina
    sabor=s;
    cant=ca; }

 //Los métodos set() y get() quedan a cargo del alumno.

 public String toString()
 { return "Caramelo:" + super.toString() + "Sabor:" + sabor + "Cantidad:"+ cant; }
}
```

//CarameloRelleno.java

```
public class CarameloRelleno extends Caramelo
{ private String saborrelleno;
   private int cantidad;

  public CarameloRelleno()
  { super();
     saborrelleno="";
     cantidad=0;
   }

  public CarameloRelleno(int c, String n, float p, String i, String s, int can,String sr, int ct)
  { super(c,n,p,i,s,can);
     saborrelleno=sr;
     cantidad=ct;
   }

  public String getSaborRelleno()
  {return saborrelleno;}

  public void setSaborRelleno(String s)
  { saborrelleno=s;}

  public int getCantidad()
  {return cantidad;}

  public void setCantidad(int c)
  { cantidad=c;}

  public String toString()
  { return "Caramelo Relleno:" + super.toString() + "Sabor Relleno:" + saborrelleno + "Cantidad:"
+ cantidad;  }
}
```

//FabricaGolosina.java

```
public class FabricaGolosina
{ private Golosina g[];

  public FabricaGolosina()
  {g=new Golosina[50];}

  public FabricaGolosina(int cant)
  { g=new Golosina[cant]; }

 public void setGolosina(int indice,Golosina golo)
  { g[indice]=golo; }

  public Golosina getGolosina(int indice)
  { return g[indice];}

  public String SaborRelleno()
  { String aux="";
     for(int i=0; i<g.length;i++)
       if (g[i]!= null && (g[i] instanceof CarameloRelleno))
       {
             CarameloRelleno cr = (CarameloRelleno) g[i];
             aux+= cr.getSaborRelleno();

       }
     return aux;
  }
}
```

//Aplicacion.java

```
public class Aplicacion
{ public static void main(String args[])
  { int c,co,ca,can,op;
```

```
     float p;
     String n,s,sr,ing;

     FabricaGolosina fg;
     Caramelo car;
     CarameloRelleno cr;

     System.out.println("Ingrese cantidad de golosinas:");
     c=In.readInt();
     fg=new FabricaGolosina(c);

     for(int i=0;i<c;i++)
     { System.out.println("Opcion Golosina:1-Caramelo 2-Caramelo Relleno:");
        op=In.readInt();

        System.out.println("Ingrese codigo:");
        co=In.readInt();
        System.out.println("Ingrese Nombre:");
        n=In.readString();
        System.out.println("Ingrese precio:");
        p=In.readFloat();
        System.out.println("Ingrese ingrediente:");
        ing=In.readString();
             System.out.println("Ingrese sabor:");
        s=In.readString();
        System.out.println("Ingrese cantidad:");
        ca=In.readInt();

        if(op==1)
        { car=new Caramelo(co,n,p,ing,s,ca);
           fg.setGolosina(i,car);
        }
       else
       { System.out.println("Ingrese sabor relleno:");
          sr=In.readString();
          System.out.println("Ingrese cantidad relleno:");
          can=In.readInt();
          cr=new CarameloRelleno(co, n, p, ing, s, ca, sr, can);
          fg.setGolosina(i,cr);
        }
    }

     //Muestro el sabor de relleno
   System.out.println("Los sabores son:"+fg.SaborRelleno());}
}//Fin Aplicación.
```

3**.** Relevando una Juguetería de la ciudad surgieron las siguientes entidades:

Juguete: Que contiene un nombre, origen (1- nacional, 2-importado), edad recomendada (1 - 2 a 4 años, 2 -5 a 9 años, 3 - 10 años en adelante), tipo (1 -Peluche 2- Rompecabezas). También posee un método que retorna el precio, que será calculado según el tipo de juguete. No se harán instancias de esta clase.

Rompecabezas: Es un Juguete. Tiene Cantidad de piezas (1- 500, 2-1000, 3-3000). Su precio se calcula teniendo en cuenta que:
500 piezas: $15,50 , 1000 piezas: $40, 3000 piezas: $49,90. Además, si el producto es nacional se le agregan $25 al precio final, y si es Importado, $100.

Peluche: Es un juguete. Tiene 2 atributos: Tamaño (1-Chico, 2-Mediano, 3-Grande) y Novedad (1-Si, 2-No), que

indica si es nuevo o no. Su precio se calcula teniendo en cuenta que:

Chico: $10, Mediano: $37,50 , Grande: $100, 90. Si el Peluche es una novedad, su precio final sube $50.

ListaJuguetes: Es un arreglo que almacena hasta 100 juguetes.

Se pide:

1) Cargar un juguete.
2) Borrar Juguete.
3) Mostrar los datos de todos los juguetes existente.
4) Mostrar los datos del peluche mas caro.
5) Informar si existen mas Peluches nacionales que Rompecabezas importados.
6) Modificar el origen de un juguete y mostrar su nuevo precio.

Resolución.

```
//Juguete.java
public abstract class Juguete
{
private String nombre;
private int origen;
private int tipo;
private int edad;
public Juguete (String n,int o,int t,int er)
{
nombre=n;
origen=o;
tipo=t;
edad=er;
}
public void setNombre (String x)
{nombre=x;}

public void setOrigen (int x)
{origen=x;}

public void setTipo (int x)
{tipo=x;}

public void setEdad (int x)
{edad=x;}

public String getNombre()
{return nombre;}

public int getOrigen ()
{return origen;}

public int getEdad()
{return edad;}

public int getTipo ()
{return tipo;}

//Método abstracto..No tiene implementación a este nivel.
public abstract float getPrecio ();

public String toString()
{
String aux="",aux2="";
switch (origen)
{
```

```
case 1:aux="Nacional ";break;
case 2:aux="Importado";break;
}
switch (edad)
{
case 1:aux2="2-4 ";break;
case 2:aux2="5-9 ";break;
case 3:aux2="10 en adelante";break;
}
return "\nNombre: "+nombre+"\nOrigen: "+aux+"\nTipo: "+tipo+"\nEdad Recomendada: "+aux2;
}
}

//Rompecabezas.java
public class Rompecabezas extends Juguete
{
private int cantPiezas;
public Rompecabezas (String n,int o,int t, int er,int cant)
{
super (n,o,t,er);
cantPiezas=cant;
}
public void setCantPiezas(int x)
{cantPiezas=x;}
public int getCantPiezas ()
{return cantPiezas;}
public float getPrecio ()
{
float p=0;
switch (cantPiezas)
{
case 1:p=(float)15.50; break;
case 2:p=40;break;
case 3:p=(float)149.90;break;
}
if(super.getOrigen()==1)
return (p+25);
else
return (p+100);
}
public String toString()
{
String aux="";
switch (cantPiezas)
{
case 1:aux="500";break;
case 2:aux="1000";break;
case 3:aux="3000";break;
}
return super.toString()+"\nCantidad de piezas: "+aux+"\nPrecio: "+this.getPrecio();
}
}
//Peluche
public class Peluche extends Juguete
{
private int tamanio;
private int novedad;
public Peluche (String n,int o,int t, int er,int tam,int nov)
{
super (n,o,t,er);
tamanio=tam;
novedad=nov;
}
public void setTamanio(int x)
{tamanio=x;}
public int getTamanio ()
{return tamanio;}
```

```
public void setNovedad (int x)
{novedad=x;}
public int getNovedad()
{return novedad;}

public float getPrecio ()
{
float p=0;
switch (tamanio)
{
case 1:p=10; break;
case 2:p=(float)37.50;break;
case 3:p=(float)100.90;break;
}
if(novedad==1)
return (p+50);
else
return p;
}

public String toString()
{
String aux="",aux2="";
switch (tamanio)
{
case 1:aux="Chico";break;
case 2:aux="Mediano";break;
case 3:aux="Grande";break;
}
if (novedad==1)
aux2="Si";
else
aux2="No";
return super.toString()+"\nTamaño: "+aux+"\nNovedad: "+aux2+"\nPrecio: "+this.getPrecio();
}
}
//ListaJuguetes.java
public class ListaJuguetes
{
private Juguete lj [];
public ListaJuguetes ()
{
lj=new Juguete [100];
}
//------------------Agregar juguete
public void setListaJuguete (int i, Juguete x)
{lj[i]=x;}
//------------------Mostrar juguete
public Juguete getListaJuguete (int i)
{return lj[i];}
//------------------Mostrar todos losjuguetes
public String toString ()
{
String aux="";
for(int i=0;i<lj.length;i++)
{if(lj[i]!=null)
       aux+=lj[i].toString()+"\n-----------------------";
}
return aux;
}
//------------------borrar juguete
public String borrarJuguete (String n)
{
for(int i=0;i<lj.length;i++)
{
if(lj[i]!=null && lj[i].getNombre().compareTo(n)==0)
       {lj[i]=null;
```

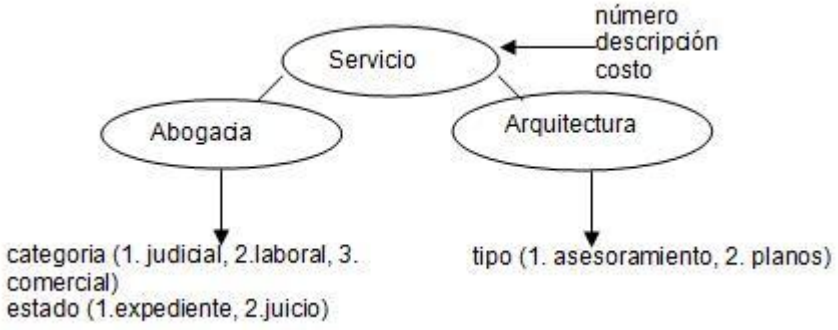
```
       return "Juguete borrado con exito";
       }
}
return "No se encontro un juguete con ese nombre";
}
//------------------Retornar Juguete mas Caro
public Juguete JugueteMasCaro ()
{
int b=0;
Juguete may=null;
for (int i=0;i<lj.length;i++)
{
if (lj[i]!=null && lj[i].getTipo()==2)
       {
       if(b==0||lj[i].getPrecio()>may.getPrecio())
              {may=lj[i];
               b=1;}
       }

}
return may;
}
//------------------Informar si existen mas Peluches nacionales o mas rompecabezas importados
public String Comparar ()
{
String aux="";
int cont1=0,cont2=0;
for (int i=0;i<lj.length;i++)
{
if(lj[i]!=null)
       {
       if(lj[i].getTipo()==2 && lj[i].getOrigen()==1)
              cont1++;
       if(lj[i].getTipo()==1 && lj[i].getOrigen()==2)
              cont2++;
       }
}
if(cont1==cont2)
aux="Hay cantidades iguales";
else
       {
       if(cont1>cont2)
       aux="Hay mas Peluches nacionales";
       else
       aux="Hay mas Rompecabezas importados";
       }
return aux;
}
//------------------modificar el origen de un juguete y Mostrar cual es su nuevo precio
public String modificarOrigen (String nom)
{
int no=0;
for (int i=0;i<lj.length;i++)
{
if(lj[i]!=null&&lj[i].getNombre().compareTo(nom)==0)
       {
       System.out.println("Datos del juguete"+lj[i].toString());
       do{
       System.out.println("\nIngrese un nuevo Origen (1-Nacional 2-Importado)");
       no=In.readInt();
       }
       while (no!=1&&no!=2);
       lj[i].setOrigen(no);
       return "Su nuevo precio es: "+lj[i].getPrecio();
       }
```

```
}
return "No se encontro el juguete";
}
}
//Juguetería.java
public class Jugueteria
{
public static void main (String args [])
{int cant=0;
ListaJuguetes lista=new ListaJuguetes();
char op;
do{
System.out.println("\nMenu de opciones");
System.out.println("\n1)Cargar Juguete \n2)Borrar Juguete \n3)Mostrar los datos de todos los
juguetes existente");
System.out.println("4)Mostrar los datos del peluche mas caro \n5)Informar si existen mas Peluches
nacionales que rompecabezas importados");
System.out.println("6)Modificar el origen de un juguete y mostrar su nuevo precio\n7)Salir");
System.out.println("Ingrese una opcion: ");

op=In.readChar();
switch (op)
{
case '1':
if(cant==100)
System.out.println("No hay mas lugar para cargar Juguetes");
else
{
Juguete aux;
String n;
int or,t,e;
System.out.println("Ingrese los datos del juguete a cargar:");
System.out.println("\nNombre: ");
n=In.readLine();
do
{
System.out.println("\nOrigen: (1-Nacional, 2-importado: )");
or=In.readInt();
}
while (or!=1&&or!=2);
do
{
System.out.println("\nEdad recomendada: (1)- 2 a 4 , 2)- 5 a 9, 3)- 10 en adelante )");
e=In.readInt();
}
while (e<1||e>3);
do
{
System.out.println("\nTipo: (1- Nacional , 2-Importado)");
t=In.readInt();
}
while (t!=1&&t!=2);
      if(t==1)
      {
      int ca;
      do
      {
      System.out.println("\nCantidad de Piezas: (1- 500 , 2- 1000, 3- 3000 )");
      ca=In.readInt();
      }
      while (ca<1||ca>3);
      aux=new Rompecabezas (n,or,t,e,ca);
      }
      else
      {
      int ta,nov;
      do
```

```
        {
        System.out.println("\nTamaño: (1- Chico , 2- Mediano, 3- Grande )");
        ta=In.readInt();
        }
        while (ta<1||ta>3);
        do
        {
        System.out.println("\nEs novedad 1-Si , 2- NO)");
        nov=In.readInt();
        }
        while (nov!=1&&nov!=2);
        aux=new Peluche (n,or,t,e,ta,nov);
        }
        lista.setListaJuguete(cant,aux);
        cant++;
}
break;
case '2':
if(cant==0)
System.out.println("NO hay Juguetes para borrar");
else
{
String nom;
System.out.println("\nNombre del jugute a borrar: ");
nom=In.readLine();
System.out.println(lista.borrarJuguete(nom));
cant--;
}
break;
case '3':
if(cant==0)
System.out.println("No hay Juguetes cargados");
else
{
System.out.println("Los Juguetes cargados son: "+lista.toString());
}
break;
case '4':
if(cant==0)
System.out.println("No hay Juguetes cargados");
else
{
        System.out.println("El juguete mas caro es: "+lista.JugueteMasCaro().toString());

}
break;
case '5':
if(cant==0)
System.out.println("No hay Juguetes cargados");
else
{System.out.println(lista.Comparar());}
break;
case '6':
if(cant==0)
System.out.println("No hay Juguetes cargados");
else
{
String n;
System.out.println("\nNombre del juguete a buscar");
n=In.readLine();
System.out.println(lista.modificarOrigen(n));
}
break;
}

}
```

```
while (op!='7');
//cargar juguete, borrar juguete,mostrar los datos del Peluche mas caro,
//Informar si existen mas Peluches nacionales o mas rompecabezas importados
//mostrar los datos de todos los juguetes existentes, modificar el origen de un juguete y cual es
su nuevo precioo

}
}
```

4. Una consultora brinda distintos servicios entre ellos la consultoría de abogados y de arquitectos, los datos utilizados son los siguientes:



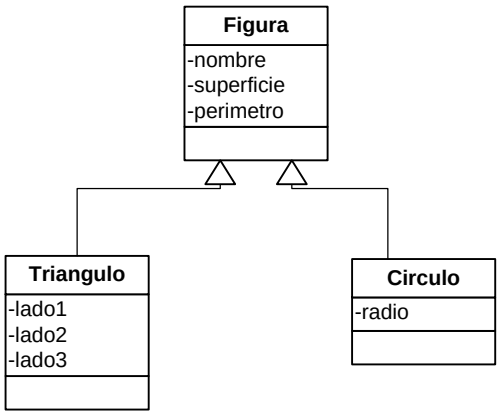
Nota: El costo del servicio de abogacía se calcula multiplicando el costo por un importe fijo que varía para expediente es 100$ y para juicio es 200$.
Para el costo del servicio de arquitectura solo se agrega un importe de 200$ si el servicio es de tipo planos.
Se pide:

1. Las tres clases, todas deben tener constructor con y sin argumentos, métodos get y set, toString() y getCosto() retorna el costo del servicio (considerar la nota)

2. Desarrollar un programa servidor que reciba los servicios enviados por el cliente, los almacene en un vector de servicios y mostrar el costo de cada servicio.

3. Totalizar el costo de los servicios de abogacía.

5. Considerando la siguiente jerarquía de clases, implementar en Java la misma.



Se pide:

a. Implementar los metodos get() , set() , toString() correspondiente. Además de los métodos CalcularPerimetro() y CalcularSuperficie().

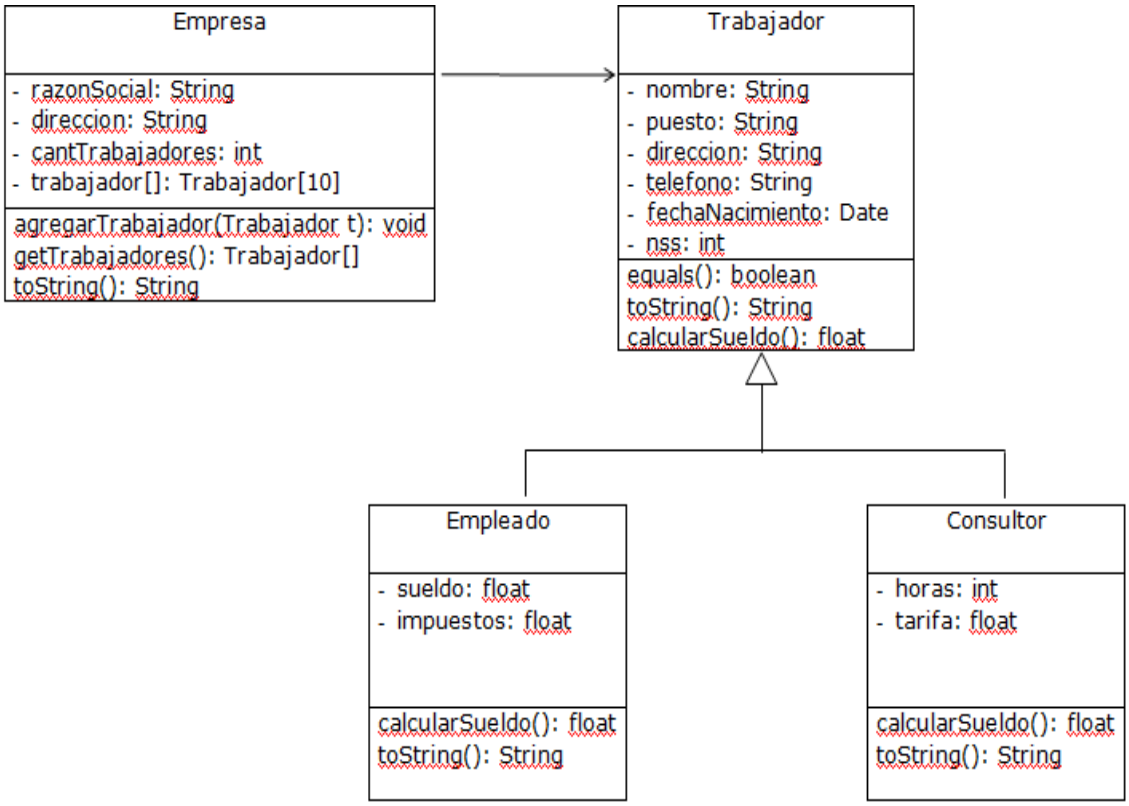
b. Almacenar en un vector todas las figuras ya sea circulo o triangulo.

c. Mostrar el valor acumulado de la superficie de todas las figuras (ya sea triangulo o circulo).

d. Mostrar el nombre del triangulo que mayor perímetro tiene.

6. Implementar las clases que se detalla a continuación de manera que permita el cálculo del sueldo según su condición (Empleado o Consultor).

A continuación se presenta el esquema de clases, y se deben tener las siguientes consideraciones:



- Los impuestos del empleado corresponden al 3% de su sueldo.
- El calculo del sueldo varia según el tipo de trabajador:
  - Para el empleado se calcula obteniendo la diferencia entre el sueldo y los impuestos, dividiendo el resto por una constante correspondiente a la nomina denominada PAGAS (=14).
  - Para el Consultor se calcula multiplicando la cantidad de horas de trabajo por la tarifa (valor de la hora de trabajo del consultor).

a) Desarrollar las clases descritas anteriormente con los métodos correspondientes.
b) Desarrollar una clase Main donde se manifieste el uso de todas las clases y métodos desarrollados.

BIBLIOGRAFÍA

Aprenda Java como si estuviera en primero. Marzo 1999 Autores: Javier García de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazales, Alberto Larzabal, Jesús Calleja, Jon García.

Notas de Clases de la cátedra Paradigmas de Programación confeccionada por la Ing. Analia Guzmán.