LABORATORIO DE PROGRAMACIÓN ORIENTADA A OBJETOS

DRA. MARICELA BRAVO

DIVISION DE CIENCIAS BASICAS E INGENIERIA
UAM - Azcapotzalco

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco

PRINCIPIO DE POLIMORFISMO

2

POLIMORFISMO

- Capacidad que permite a dos ***clases diferentes*** responder de forma distinta a un ***mismo mensaje***
- Esto significa que dos clases que tengan un método con el mismo nombre y que respondan al mismo tipo de mensaje (es decir, que reciban los mismo parámetros), ejecutarán acciones distintas.
- Clase abstracta
- Interfaces

3

CLASE ABSTRACTA

- Una **clase abstracta** es una clase de la que no se pueden crear objetos, pero puede ser utilizada como clase padre para otras clases.
- Declaración:

```
abstract class NombreClase {
    ..............
}
```

4

CLASE ABSTRACTA

Es una clase que declara la existencia de métodos pero no la implementación de dichos métodos.

Una clase abstracta puede contener métodos no-abstractos pero al menos uno de los métodos debe ser declarado abstracto.

```
abstract class Drawing
{
  abstract void miMetodo(int var1, int var2);
  String miOtroMetodo( ){ ... }
}
```

Una clase abstracta no se puede instanciar pero si se puede heredar y las clases hijas serán las encargadas de agregar la funcionalidad a los métodos abstractos.

Si no lo hacen así, las clases hijas deben ser también abstractas.

5

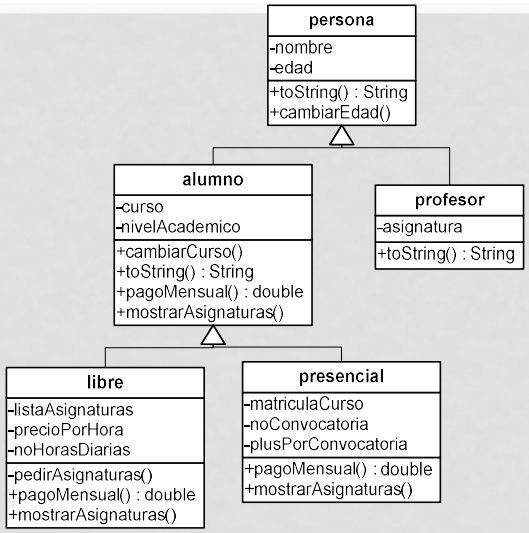
EJEMPLO 1 DE CLASE ABSTRACTA

```
abstract class ObjetoGrafico
{
  int x, y; ...
  void mueveA(int newX, int newY)
  { ... }

  abstract void dibujar();
  abstract void ajustar();
}
```

```
class Circulo extends ObjetoGrafico
{
   void dibujar() { ... }
   void ajustar() { ... }
 }

class Rectangulo extends ObjetoGrafico
{
  void dibujar() { ... }
  void ajustar() { ... }
}
```

6

EJEMPLO 2 DE CLASE ABSTRACTA



7

EJEMPLO 2 DE CLASE ABSTRACTA

```
abstract class Alumno extends Persona
{
   protected int curso;
   private String nivelAcademico;
   public Alumno (String n, int e, int c, String nivel)
   {
         super(n, e);
         curso = c;
         nivelAcademico = nivel;
   }
   public String toString()
   {
         return super.toString() + curso + nivelAcademico;
   }
   abstract double pagoMensual();
   abstract String getAsignaturas();
}
```

8

EJEMPLO 2 DE CLASE ABSTRACTA

```
class Libre extends Alumno
{
   private String []listaDeAsignaturas;
   private static float precioPorHora=10;
   private int noHorasDiarias;
   private void pedirAsignaturas() {}// se inicializa listaDeAsignaturas
   public double pagoMensual() {
            return precioPorHora*noHorasDiarias*30; }
   public String getAsignaturas() {
             String asignaturas="";
             for (int i=0; i<listaDeAsignaturas.length; i++)
                        asignaturas += listaDeAsignaturas[i] + ' ';
             return asignaturas;
   }
   public Libre(String n, int e, int c, String nivel, int horas)
    {super(n,e,c,nivel); noHorasDiarias = horas; pedirAsignaturas(); }
}
```

9

EJEMPLO 2 DE CLASE ABSTRACTA

```
class Presencial extends Alumno
{
    private double matriculaCurso;
    private double plusPorConvocatoria;
    private int noConvocatoria;

    public double pagoMensual()
    {
            return (matriculaCurso+plusPorConvocatoria*noConvocatoria)/12;
    }
    public String getAsignaturas()
    {
             return "todas las del curso " + curso;
    }
    public Presencial(String n, int e, int c, String nivel,double mc, double pc, int nc)
    {
             super(n,e,c,nivel);
             matriculaCurso=mc;
             plusPorConvocatoria=pc;
             noConvocatoria=nc;
    }
}
```

10

EJEMPLO DE CLASE ABSTRACTA

```
// FUNCIONES GLOBALES
void mostrarAsignaturas(Alumno v[]) {
 for (int i=0; i<v.length; i++)
  System.out.println(v[i].getAsignaturas());
  // enlace dinámico
}
double totalPagos(Alumno v[]) {
 double t=0;
 for (int i=0; i<v.length; i++)
  t += v[i].pagoMensual(); // enlace dinámico
 return t;
}
```

11

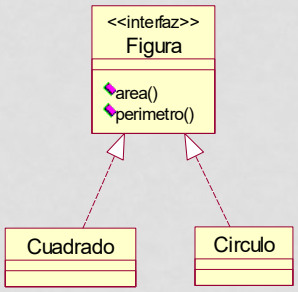
INTERFACES

- Podría suceder que los objetos de varias clases compartan la capacidad de ejecutar un conjunto de operaciones.
- Y dependiendo de la clase de objeto, cada operación se realice de diferente manera.
- Ejemplo:
  - Clases: Circulo, Elipse, Triangulo, ...
  - Todas esas clases incluyen los métodos: área, perímetro, cambiar escala, etc.
- Podríamos definir una interface común que agrupe todos los métodos comunes (como métodos abstractos).
- Y luego definir varias clases de modo que implementen una misma interfaz.

12

EJEMPLO 1 DE INTERFACE

```
public interface Figura {
   abstract double area();
   abstract double perimetro();
}
public class Circulo implements Figura {
   private double radio;
   private static double PI=3.1416;
   ..............
   public double area() { return
   PI*radio*radio; }
   public double perimetro() { return
   2*PI*radio; }
}
public class Cuadrado implements Figura {
   private double lado;
   ..............
   public double area() { return lado*lado; }
   public double perimetro() { return 4*lado; }
}
```



13

EJEMPLO DE INTERFACE

Una interface puede incluir también definiciones de constantes aparte de métodos abstractos.

Una misma clase puede implementar más de una interface ⇒ Herencia múltiple de interfaces

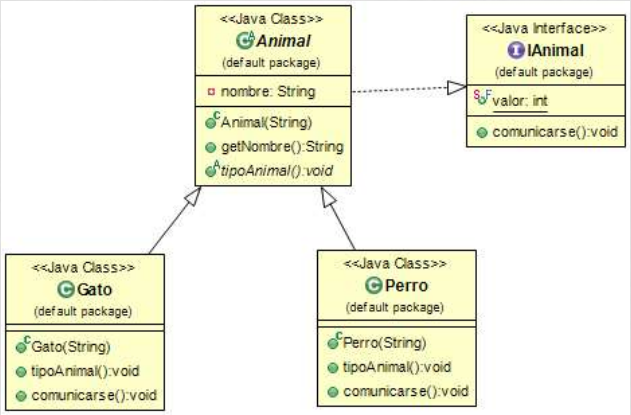
Se pueden crear jerarquías de interfaces (con extends!!).

Se pueden declarar referencias a objetos que implementen una cierta interface.

```
double totalArea(Figura v[]) {
 double t=0;
 for (int i=0; i<v.length; i++)
  t += v[i].area();
 return t;
}
```

14

EJEMPLO 2



15

EJEMPLO 2

```
public interface IAnimal
{

  int valor=5;

  /* Método Comunicarse, será implementado por las
clases concretas que hereden de la clase Animal
   */

   public void comunicarse();

}
```

16

EJEMPLO 2

```
public abstract class Animal implements IAnimal
{
 private String nombre;
public Animal (String nombre)
 {
  this.nombre=nombre;
  System.out.println("Constructor Animal, " +
    "nombre del animal : "+this.nombre);
 }

 public String getNombre()
 {
  return nombre;
 }
public abstract void tipoAnimal();

}
```

17

CLASES ABSTRACTAS VS INTERFACES

- Las clases abstractas son similares a las interfaces en lo siguiente:
  - No se pueden instanciar objetos de ellas
  - Pueden contener una mezcla de métodos abstractos y métodos con implementación.

18

CLASES ABSTRACTAS VS INTERFACES

- Con las clases abstractas
  - Es posible declarar atributos que no son static y final
  - Es posible declarar métodos public, protected, y private.
- Con las Interfaces
  - Todos los atributos son automáticamente public, static y final.
  - Todos los métodos son public.
  - Solamente se puede extender una clase, sea o no abstracta.
  - Es posible implementar cualquier número de interfaces.

19

CUANDO UTILIZAR CLASES ABSTRACTAS O INTERFACES

- Considera utilizar clases abstractas cuando:
  - Se desea compartir código entre varias clases estrechamente relacionadas.
  - Se espera que las clases que extienden de la clase abstracta posean muchos métodos comunes, o requieran del uso de modificadores de acceso diferentes a public (protected y private).
  - Se necesita declarar atributos non-static y non-final. Esto permite definir métodos que pueden accesar y modificar el estado del objeto al cual pertenecen.

20

CUANDO UTILIZAR CLASES ABSTRACTAS O INTERFACES

- Considerar el uso de Interfaces cuando:
  - Se espera que clases no relacionadas implementen la Interface.
  - Se necesita especificar el comportamiento de un tipo de dato particular, independientemente de quien implementa su comportamiento.

21

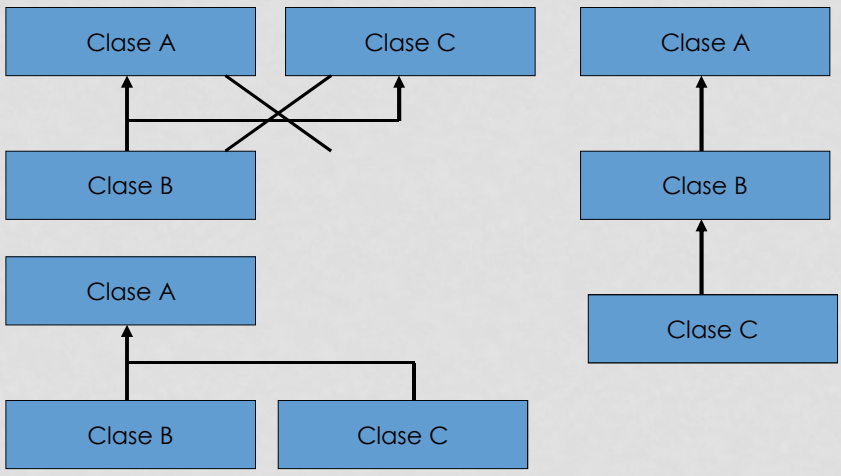
HERENCIA MULTIPLE

22

HERENCIA MÚLTIPLE

- Reglas básicas para la herencia en Java
  - No está permitida la herencia múltiple
  - Si es posible la herencia multinivel, A puede ser heredada por B y C puede heredar de B.
  - Una clase puede ser heredada por varias clases.

HERENCIA MULTIPLE



HERENCIA MÚLTIPLE

- Cuando se trabaja con clases en Java se está limitado a heredar solamente de una clase base.
- Las interfaces relajan esta restricción al permitir realizar herencia múltiple mediante la combinación de varias interfaces en una misma clase.
- Para hacer esto, solo se colocan los nombres de las interfaces después de la palabra reservada implements separadas por comas.

25

EJEMPLO 1: EXTENDS - IMPLEMENTS

```
interface Hello1 // 1st interface
{
public abstract void display1();
}

interface Hello2 // 2nd interface
{
 public abstract void display2();
}

class Test1 // concrete (non-abstract) class
{
  public void calculate()
{
  System.out.println("Concrete class Test1 method calculate()
    executed");
}
}
```

26

EJEMPLO 1: EXTENDS - IMPLEMENTS

```
abstract class Test2 extends Test1 // abstract class extending Test1
{ public abstract void show(); }
public class Demo extends Test2 implements Hello1, Hello2
{
public void display1()
{ System.out.println("Interface Hello1 method display1() executed"); }
public void display2()
{ System.out.println("Interface Hello2 method display2() executed"); }
public void show()
{ System.out.println("Abstract class Test2 method show() executed");
}
public static void main(String args[])
{
Demo d1 = new Demo();
d1.display1();
d1.display2();
d1.calculate(); d1.show();
}
}
```

27

EJEMPLO 2: EXTENDS - IMPLEMENTS

```
interface Forward { void drive(); }
interface Stop { void park(); }
interface Speed { void turbo(); }

class GearBox { public void move() { } }

class Automatic extends GearBox implements
Forward, Stop, Speed
{
    public void drive() { System.out.println("drive()"); }
    public void park() {  System.out.println("park()");}
    public void turbo() {  System.out.println("turbo()"); }
    public void move() { System.out.println("move()"); }
}
```

28

EJEMPLO 2: EXTENDS - IMPLEMENTS

```
public class Car {
        public static void cruise(Forward x) { x.drive(); }
        public static void park(Stop x) { x.park(); }
        public static void race(Speed x) { x.turbo(); }
        public static void move(GearBox x) { x.move(); }

        public static void main(String[] args)
        {
            Automatic auto = new Automatic();
            cruise(auto); // Interface Forward
            park(auto); // Interface Stop
            race(auto); // Interface Speed
            move(auto); // class GearBox
        }
    }
```

29

EJEMPLO 3: SUPER HEORES

- Clases abstractas
- Interfaces

30

EJEMEPLO 3: EXTENDS - IMPLEMENTS

```
interface SuperHero {
    void powers();
}
interface Alien {
    void planet();
}
interface SuperMan extends  Alien, SuperHero {
    void xRayVision();

}
interface Human {
    void normal();
}
interface Monster extends Alien {
    void killHuman();
}
```

31

EJEMPLO 3: EXTENDS - IMPLEMENTS

```
class Skrull implements  Monster {
    void shapeShift() {}
    public void killHuman() {}
    public void planet() {}
}

class ClarkKent implements SuperMan, Human {
    public void normal() {}
    public void planet() {}
    public void xRayVision() {}
    public void powers() {}
}

class Hulk implements Human, SuperHero {
    void thunderClap() {};
    public void normal() {}
    public void powers() {}
}
```

32

EJEMPLO 3: EXTENDS - IMPLEMENTS

```
public class Earth
{
    static Alien invasion(Monster mt) {
        return mt;
    }

    static SuperMan change(ClarkKent ck) {
        return ck;
    }

    static void battle(SuperHero sh, Monster m) {
        sh.powers();
        m.planet();
        m.killHuman();
    }

    public static void main(String[] args) {
        Skrull ogre = new Skrull();
        invasion(ogre);
        ClarkKent christopherReeve = new ClarkKent();
        SuperMan superMan = change(christopherReeve);
        battle(superMan, ogre);
        battle(new Hulk(), ogre);
    }
}
```

33

DUDAS O COMENTARIOS???