Chuletario de OO en Java

CLASE	**Constructor Por defecto**	**Constructor Copia**	**Constructor Parametrizado**	**get ()**	**set (valuex)**	**toString ()**	**equals (Object o)**	**Ejemplos**
	Objetivo: Inicializar por defecto.	Objetivo: Hacer una Copia del objeto recibido (misma clase).	Objetivo: Inicializar a los valores recibidos	Objetivo: Devolver Nota: Existe 1 get por cada atributo.	Objetivo: Asignación Nota: Existe 1 set por cada atributo.	Objetivo: return String	Objetivo: Redefinir el método equals de la clase Object --return boolean--	S.o.p=System.out.println.
Simple Ejemplo: class Persona { **private int x;** **private float y;**	[Sin valores previos recibidos] public Persona(){ x = 0; y = 0.0; }	[Valor recibido: Objeto de la misma clase] public Persona(Persona objeto){ x = objeto.x; y = objeto.y; }	public Persona(int valuex, float valuey){ this.x=valuex; this.y=valuey; }	public int getX () { return x; }	public void setX (int valuex) { this.x=valuex; }	public String toString() { String temp=”Valor de x=“+ x + “ y valor de y= ”+ y; return temp; }	public boolean equals (Object o) { //1º) Upcasting a la CD Persona p = (Persona) o; return this.x==p.x && this.y==p.y; }	Persona p1= new Persona();//CD Persona p2= new Persona(4,5.0); Persona p3=new Persona (p2); S.o.P(p1.getX()); p2.setX(8); if (p1.equals(p2)) ….. else …. S.o.p(p1.toString());
Objeto String Ejemplo: class Persona{ private int x; **private String nombre;**	[Sin valores previos recibidos] public Persona(){ x = 0; nombre =new String (); }	[Valor recibido: Objeto de la misma clase] public Persona(Persona obj){ x = obj.x; nombre = new String (obj.nombre) ; // o //nombre=obj.nombre; //idem }	[Valores recibidos: x, nombre] public Persona(int valuex, String valuenombre){ this.x=valuex; this.nombre=valuenombre; //this.nombre=new String (valuenombre); //idem }	public int getX () { return x; } public String getNombre () { return nombre; }	public void setX (int valuex) { this.x=valuex; } public void setNombre (String valuenombre) { this.nombre=valuenombre; }	public String toString() { String temp=”X vale =” + x + “y Nombre = “ + nombre; return temp; }	public boolean equals (Object o) { //1º) Upcasting a la CD Persona p = (Persona) o; return this.x==p.x && this.nombre.**equals** (p.nombre); }	Persona p1 = new Persona(); Persona p2 = new Persona (p1); Persona p3 = new Persona (15,”J”); S.o.p(p1.getX() + ' y ' + p1.getNombre()); S.o.p(p1.toString()); p2.setNombre(“Pepe”); if (p1.equals(p2)) …. else ….
Objeto: Composición (**relación → tiene**) Ejemplo: class Casa{ private int num; **private Persona propietario;**	[Sin valores previos recibidos] public Casa(){ num = -1; propietario =new Persona (); }	[Valor recibido: Objeto de la misma clase] public Casa (Casa obj){ num = obj.num; propietario = new Persona (obj.propietario) ; //propietario=obj.propietario; }	[Valores recibidos: x, objeto] public Casa (int valuex, Persona p){ this.num=valuex; this.propietario=p; //this.propietario=new Persona (p); //idem }	public int getNum () { return num; } public Persona getPropietario () { return propietario; }	public void setNum (int valuex) { this.num=valuex; } public void setPropietario (Persona objeto) { this.propietario=objeto; }	public String toString() { String temp=”Vivo en la casa nº “ + num + “y soy = “ + propietario.toString(); //propietario.getNombre(); return temp; }	public boolean equals (Object o) { //1º) Upcasting a la CD Casa c = (Casa) o; return this.num==c.num && this.prorietario.**equals** (c.propietario); }	Casa c1 = new Casa(); //CD Casa c2 = new Casa(c1); //CC Casa c3= new Casa(38,new Persona (36,'Luis')); Persona p= new Persona(36,”Luis”) Casa c4=new Casa (38, p); S.o.p(c3.getPropietario().toString()); if (c1.equals(c2)) …. else ….
Objeto: Composición * (**relación → tiene varios**) Ejemplo: class Casa{ private int num; **private Persona[] habitantes;** private int tamanyo;	[Sin valores previos recibidos] public Casa(){ num = -1; tamanyo=10; habitantes=new Persona[tamanyo]; }	[Valor recibido: Objeto de la misma clase] public Casa (Casa obj){ num = objeto.num; tamanyo=objeto.tamanyo; habitantes=new Persona[objeto.tamanyo]; //Copiar los datos habitantes= Arrays.copyOf (obj.habitantes,obj.length); }	[Valores recibidos: x, objeto] public Casa (int t){ num = -1; tamanyo=t; habitantes=new Persona[tamanyo]; }	public Persona[] getHab(){ return habitantes; }		public String toString() { String temp=”Vivo en la casa nº “ + num + “y somos= “ + habitantes.toString() ; return temp; }	public boolean equals (Object o) { //1º) Upcasting a la CD Casa c = (Casa) o; return this.num==c.num && **Arrays.equals(habitantes, c.habitantes)** }	**Nota:** Debe existir un método para **agregar, eliminar y recuperar** objetos a la colección public void agregar (Persona p, int pos){ habitantes[pos]=p; } public Persona **recuperar(int pos){** return habitantes[pos]; **}** public void **eliminar(int pos){** habitantes[pos]=null; **}**

Chuletario de OO en Java

CLASE	**Constructor Por defecto**	**Constructor Copia**	**Const. Parametrizado**	**get ()**	**set (valuex)**	**toString ()**	**equals (Object o)**	**Ejemplos**
Objeto: Composición * (**relación → tiene varios**) Ejemplo: class Casa{ private int num; **private List habitantes;**	[Sin valores previos recibidos] public Casa(){ num = -1; habitantes=new ArrayList(); }	[Valor recibido: Objeto de la misma clase] public Casa (Casa obj){ num = objeto.num; habitantes=new ArrayList();; Colllections.copy (obj.habitantes, habitantes); }	[Valores recibidos: x, objeto] public Casa (int n){ num = n; habitantes=new ArrayList(); }	public List getHab(){ return habitantes; }	public void setHab(List l){ habitantes=l; }	public String toString() { String temp=”Vivo en la casa nº “ + num + “y somos= “ + habitantes.toString() ; return temp; }	public boolean equals (Object o) { //1º) Upcasting a la CD Casa c = (Casa) o; return this.num==c.num && **habitantes.equals(c.habitantes)** }	**Nota:** Debe existir un método para **agregar, eliminar y recuperar** objetos a la colección public void **agregar** (Persona p){ habitantes.add(p); } public void **eliminar(int pos){** habitantes.remove(pos); }
Herencia 1: extends Heredan atributos y métodos protegidos Ejemplo: class Alumno extends Persona { private int curso; }	public Alumno () { //1º) Inicializar CB. **super**(); //2º) Inicializar CD. curso = 1; }	public Alumno (Alumno a) { //1º) Inicializar CB. //Opcion a. //CParametrizado de Persona: **super**(a.nombre, a.dni,a.edad); //Opcion B //Constructor Copia de Persona super(a); //2º) Inicializar CD. curso = a.curso; }	//Constructor A public Alumno (Persona p, int c) { //1º) Inicializar CB. **super**(p); //2º) Inicializar CD. curso = c; } //Constructor B public Alumno (String n, String dni, String edad, int c) { //1º) Inicializar CB. **super**(n,dni, edad); //2º) Inicializar CD. curso = c; }	**Nota**: getX() y getNombre() se heredan de la CB public int getCurso () { return (this.curso); }	**Nota**: setX() y setNombre() se heredan de la CD. public void setCurso (int valuec) { this.curso=valuec; }	public String toString () { //Opción a: return this.x + this.nombre+ this.dni+ this.edad+this.curso; //Opción b: utilizando el método toString de persona. return super.toString() +this.curso }	public boolean equals (Object o) { //1º) Upcasting a la CD Alumno a = (Alumno) o; //opción a. A nivel de atribut return this.curso==a.curso && this.nombre.equals(a.nombre) && this.dni.equals(a.dni) && this.edad==a.edad; //opción b return this.curso==a.curso && super.equals(a); }	Persona p1 = new Alumno(); Persona p2= new Alumno (10,”Luis”, 1); s.o.p(p1.getNombre()); if (p1.equals(p2)) …. else …. **//CUIDADO:ERROR** **s.o.p(p1.getCurso()); //p1 es una persona, no un alumno, no tiene acceso a .getCurso())** **Solución** **Alumno a=(Persona) p1;** **s.o.p(a.getCurso());**
Herencia 2: abstract 1. CB no puede ser instanciada. 2. CB es abstracta 3. CB puede tener métodos abstractos → Obliga a las CD a implementarlos Ejemplo: public **abstract** class Persona{ public **abstract** int XXX(); } public class Alumno { public int XXX(){ <implementación>} };								
Herencia 3: interface Establece comportamiento. 1. No tiene atributos (sólo final y static). 2. Cjto de métodos sin implementación → Obliga a las CD(implementadoras) a su codificación. 3. No tiene constructores.								Ejemplo: //Def. De la CB: public interface Figura { public int area(); } public class Circulo implements Figura { public int area () { return …..... }