1. Obreros

Dada la siguiente jerarquía de clases:

```
public interface Montador{
        void montar(String x);
        void desmontar(String x);
}
public class Obrero{
    public Obrero(){System.out.println("Se crea Obrero");}
    public void saludar(){System.out.println("Hola, soy Obrero");}
    ...
}
public class Carpintero extends Obrero implements Montador {
    public Carpintero(){System.out.println("Se crea Carpintero");}
    public void montar(String x) {System.out.println("Montando " + x);}
    public void desmontar(String x) {System.out.println("Desmontando " + x);}
    public void clavar() { … }
}
public class Albañil extends Obrero {
                public Albañil() {
                        super();
                        System.out.println("Se crea Albañil");
                }
                public void levantarMuro(){
                        System.out.println("Levantando muro …");
                }
}
```

a) Indicar qué líneas del siguiente fragmento de programa producirán errores de compilación, y

```
public static void main(String a[]){
    Montador m1 = new Carpintero();
    Montador m2 = new Albañil();
    Obrero o1 = new Carpintero();
    Obrero o2 = new Albañil();
    o1.montar("Mesa");
    o2.levantarMuro();
    m1.saludar();
    m1.montar("Silla");
    ((Albañil)o2).levantarMuro();
    ((Albañil)o1).levantarMuro();
}
```

b) Una vez eliminadas las líneas con error, indicar cuál sería la salida por pantalla del programa.

2. Usuarios

//Nota: sout = System.out.println

```
interface Instalador{
      public void instalarPrograma(String nombrePrograma);
}

class Usuario{
      protected String nombre;
      protected String contraseña;
      public Usuario(String n,String c){
            nombre = n;
            contraseña = c;
            sout("Se crea usuario " + nombre);
      }
      public void imprimirArchivo(String nombreArchivo){
            sout("El usuario " + nombre + " imprime " + nombreArchivo);
      }
}

class UsuarioAvanzado extends Usuario implements Instalador {

      public UsuarioAvanzado(String n, String c){
            super(n,c);
      }

      public void instalarPrograma(String nombrePrograma){
            sout("El usuario + nombre + " instala " + nombrePrograma);
      }

      @Override public void imprimirArchivo(String nombreArchivo){
            sout("El usuario AVANZADO " + nombre + " imprime " + nombreArchivo);
      }
}

class Administrador extends UsuarioAvanzado implements Instalador {

      public Administrador(String n, String c){
            super(n,c);
      }

      public void desinstalarPrograma(String nombrePrograma){
            sout("El usuario + nombre + " desinstala " + nombrePrograma);
      }
}

class Invitado extends Usuario {
      public Invitado(){
            super("invitado", "");
      }
}
```

Polimorfismo y genericidad

Basándote en los interfaces y clases anteriores, indica qué líneas del siguiente programa son correctas (marcando la columna OK), tienen errores de compilación (EC) o tienen errores de ejecución (EE). Si no tienen errores indica qué salida producen al ejecutarse el programa y si contienen errores, cuál es el motivo.

	OK	EC	EE	Salida / Por qué hay error
`Administrador a1 = new Administrador("Admin","0123");`				
`a1.instalarPrograma("Word");`				
`a1.imprimirArchivo("carta.txt");`				
`Instalador i1 = new UsuarioAvanzado("Borja","0123");`				
`i1.instalarPrograma("Word");`				
`i1.imprimirArchivo("carta.txt");`				
`Instalador i2 = new Invitado();`				
`i2.imprimirArchivo("carta.txt);`				
`Usuario u1 = new UsuarioAvanzado("Raul","0123");`				
`u1.instalarPrograma("Word");`				
`u1.imprimirArchivo("carta.txt");`				
`((UsuarioAvanzado)u1).instalarPrograma("Word");`				
`((Administrador)u1).desinstalarPrograma("Word");`				
`((UsuarioAvanzado)u1).desinstalarPrograma("Word");`				

3. Ejercicio 1.- Televisores

```
public interface Conectable {
   public contectar(String red);
}
```

```
public abstract class Televisor {
private String modelo;

   public Televisor(String modelo){
      this.modelo = modelo;
      System.out.print("Se crea Televisor " + modelo);
   }
   public Televisor(){
      this.modelo = "desconocido";
      System.out.print("Se crea Televisor " + modelo);
   }
   public abstract void resolucion();
}
```

```
public class TVTubo extends Televisor{
   public TVTubo (String modelo){
      super(modelo);
   }
   public TVTubo (){
      System.out.println("Se crea TVTubo");
   }
   public void resolucion(){
      System.out.println "405 lineas";
   }
}
-----------------------------------------------------------
public class TVPlano extends Televisor{
   public TVPlano (String modelo){
      super(modelo);
      System.out.println("Plano");
   }
   public void resolucion(){
      System.out.println "1920 x 1080";
   }
}
-----------------------------------------------------------
public class SmartTV extends TVPlano implements Conectable{
   public SmartTV (String modelo){
      super(modelo);
   }
   public void conectar(String redWifi){
      System.out.println("Conectado a " + redWifi);
   }
}
```

Polimorfismo y genericidad

Basándote en los interfaces y clases de la jerarquía Televisores, indica qué líneas del siguiente programa son correctas (marcando la columna OK), tiene errores de compilación (EC) o errores de ejecución (EE). Si no tienen errores indica qué salida producen al ejecutarse el programa y si contienen errores, cuál es el motivo.

	OK	EC	EE	Salida / Por qué hay error
`TVTubo tubo1 = new TVTubo("vega")`				
`tubo1.resolucion();`				
`Television tv1 = new TVTubo()`				
`tv1.resolucion();`				
`TVPlano tvp1 = new SmartTV("H55");`				
`tpv1.conectar("casa");`				
`tpv1.resolucion();`				
`SmartTV s1 = (SmartTV) tvp1;`				
`s1.conectar("vecino");`				
`Television tv2 = new Television();`				
`tv2.resolucion();`				