Ejercicio U5_B3A_E1:

```
class Ordenador{
   String procesador;
   int ram;
}

class Sobremesa extends Ordenador{
   String  caja;
}
class Portatil  extends Ordenador{
   double peso;
}

class App {
   public static void main(String[] args) {
      Sobremesa s1= new Sobremesa();
      Portatil p1=new Portatil();

      s1.procesador="core i5";
      s1.ram=8;
      s1.caja="micro-atx";

      p1.procesador="core i3";
      p1.ram=4;
      p1.peso=1.5;
   }
}
```

Ejercicio U5_B3A_E2:

```
class Persona{
   String dni;
   String nombre;
   String direccion;
}

class Empleado extends Persona{
   double sueldo;
}
class Cliente  extends Persona{
   double deuda;
}

class App {
   public static void main(String[] args) {
      Persona p= new Persona();
      p.dni="111x";
      p.nombre="una persona sin sublcase por ejemplo un proveedor";
      p.direccion="calle cachola";

      Empleado e=new Empleado();
      e.dni="222x";
      e.nombre="Elías";
      e.direccion="calle piojos";
      e.sueldo=1950.0;

      Cliente c=new Cliente();
```

```
        c.dni="333x";
        c.nombre="Telma";
        c.direccion="calle chupito";
        c.deuda=900.60;

        System.out.println("una persona genérica: " + p.dni+ "  "+ p.nombre + "  "+ p.direccion);
        System.out.println("un empleado: " + e.dni+ "  "+ e.nombre + "  "+ e.direccion+ "  " +
e.sueldo);
        System.out.println("una clienta: " + c.dni+ "  "+ c.nombre + "  "+ c.direccion+ "  " +
c.deuda);


    }
}
```

Ejercicio U5_B3A_E3:

Sin duda se echa de menos que la clase figura no tenga escrito un constructor y que fuera este constructor el que iniciara el color de la figura. Esto sería lo apropiado y no hacerlo así traerá problemas e incoherencias en programas grandes. Pero para hacerlo así necesitaríamos usar super(), un concepto que veremos en los siguientes boletines . Aquí resolvimos el problema a través de un método setColor() pero insistimos en que no es la solución correcta. Un ejemplo de una situación problemática en este contexto: imagina que la clase *figura* tuviera 20 atributos y que además la clase figura tuviera 30 subclases. En cada subclase (y hay 30) veríamos el mismo código iniciando los 20 atributos de la figura. El código duplicado delata un error en el diseño de las clases.

```
class Figura{
    private String color;
    void setColor(String color){
        this.color=color;
    }
    String getColor(){
        return color;
    }
}

class Cuadrado extends Figura{
    private double lado;

    Cuadrado(double lado, String color){
        this.setColor(color);
        this.lado = lado;
    }

    double getLado() {
        return lado;
    }
}


class Circulo extends Figura{
    private double radio;

    Circulo(double radio, String color){
        this.setColor(color);
        this.radio = radio;
    }

}

class App{
    public static void main(String[] args) {
```

```
        Cuadrado miCuadrado=new Cuadrado(2.5,"azul");
        System.out.println("Lado de miCuadrado: "+ miCuadrado.getLado());
        Circulo miCirculo=new Circulo(3.6,"blanco");
        System.out.println("Color de miCirculo: "+ miCirculo.getColor());

    }
}
```

Mejora en próximos boletines: Evitar que las clases bases se dediquen a inicializar la clase Base, lo que genera código duplicado.
Solución: Invocar al constructor de base desde subclase con super()

Ejercicio U5_B3A_E4:

creamos carpeta animales y en ella creamos 3 .java que describen la jerarquía

```
//Animal.java
package animales;

public class Animal {
     private int edad;
    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
}


//Gato.java
package animales;

public class Gato extends Animal {
    private boolean razaEuropea;

    public Gato(int edad,boolean razaEuropea) {
        this.razaEuropea=razaEuropea;
        if(edad>15){
            this.setEdad(15);
        }else{
            this.setEdad(edad);
        }

    }
    public boolean esRazaEuropea(){
        return razaEuropea;
    }


}

//Perro.java
package animales;

public class Perro extends Animal {
    private boolean puraRaza;

    public Perro(int edad,boolean puraRaza) {
        this.puraRaza = puraRaza;
        if(edad>15){
            this.setEdad(15);
        }else{
            this.setEdad(edad);
        }
    }
    public boolean esPuraRaza(){
        return puraRaza;
    }


}




//App.java
import animales.*;
public class App{
    public static void main(String[] args) {
       Perro canKan=new Perro(16,true);
       Gato  cati=new Gato(13,false);
```

```
    System.out.println("Edad canKan: "+ canKan.getEdad() +"   Es pura raza canKan: "+ canKan.esPuraRaza());
    System.out.println("Edad cati: "+ cati.getEdad() +"   cati es raza europea: "+ cati.esRazaEuropea());
  }
}
```

Observa que hay Código duplicado... Debemos sacar factor común y dejar que la superclase controle centralizadamente las cosas comunes a todos los animales en este caso vigilar que no tengan más de 15 años

```
//Animal.java
package animales;

public class Animal {
    private int edad;
   public int getEdad() {
      return edad;
   }

    public void setEdad(int edad) {
      if(edad>15)
                  this.edad = 15;
         else
                  this.edad=edad;
   }

}


//Gato.java
package animales;

public class Gato extends Animal {
   private boolean razaEuropea;

   public Gato(int edad,boolean razaEuropea) {
      this.razaEuropea=razaEuropea;
      this.setEdad(edad);
   }
   public boolean esRazaEuropea(){
      return razaEuropea;
   }


}

//Perro.java
package animales;

public class Perro extends Animal {
   private boolean puraRaza;

   public Perro(int edad,boolean puraRaza) {
      this.puraRaza = puraRaza;
      this.setEdad(edad);
   }
   public boolean esPuraRaza(){
      return puraRaza;
   }


}
```

y además como discutimos en el ejemplo anterior, en un caso real más complicado, lo correcto es utilizar super para invocar al constructor