El objetivo de herencia y composición es el mismo.
Cuando se desarrolla software, entendiendo por desarrollo no sólo la implementación sino también las fases previas de análisis y diseño, uno de los objetivos perseguidos, o por lo menos así debería ser, es conseguir la máxima capacidad de reutilización de las distintas funcionalidades implementadas, y por lo tanto también del código. En los entornos de programación orientados a objetos existen básicamente dos formas de conseguir esto: composición y herencia.

Composición vs Herencia.
Es un tema complejo e importante cuando se desarrollan aplicaciones grandes. En el contexto de grandes aplicaciones habrá un montón de clases que se pueden organizar en jerarquías o alternativamente con composición. Cada técnica tiene sus puntos fuertes y débiles. Hay en contextos que la necesidad de la herencia es indiscutible. Hay otros contextos en los que la aplicación de la composición también es indiscutible. Y hay en contextos que hay … ¡dudas de que usar!.

un ejemplo sencillo en el que observamos que ambas técnicas reutilizan código
En el siguiente ejemplo se reutiliza el código de la clase Animal usando Herencia para Perro y Composición para Gato. Discutir que es mejor "lo intentamos brevemente" más adelante, de momento simplemente observar que con los dos mecanismos se reutiliza código

```
//Animal.java
package animales;
public class Animal {
   private int edad;
   public Animal(int edad) {
      if (edad > 15) {
         this.edad = 15;
      } else {
         this.edad = edad;
      }
   }

   public int getEdad() {
      return edad;
   }
}
```

```
//Perro.java
package animales;

public class Perro extends Animal {
   private boolean puraRaza;

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


}
```

```
//Gato.java
package animales;
```

```
public class Gato{
   private boolean razaEuropea;
   private Animal animal;


   public Gato(int edad,boolean razaEuropea) {
      this.razaEuropea=razaEuropea;
      animal= new Animal(edad);

   }
   public boolean esRazaEuropea(){
      return razaEuropea;
   }
   public int getEdad(){
      return animal.getEdad();
   }


}
```

```
//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());
   }
}
```

Cuando usar herencia y cuando composición.

Herencia, cuando hay una relación "Es un", es decir cuando está clarísimo que la subclase es una especialización de la superclase. Por ejemplo, un gato es un animal y esto siempre es así y estamos seguros que siempre será así

Composición, cuando hay una relación "tiene" o "es parte de". Por ejemplo un neumático es parte de un coche, o bien, un coche tiene un neumático

cuando "hay dudas" se prefiere composición

Ejemplo: se detecta que la relación "Es un" no es para siempre. Supongamos que un "un empleado es un trabajador" y "un autónomo es un trabajador". Pero por lo que sea un trabajador puede cambiar su relación con la empresa y pasar por ejemplo de empleado a autónomo(los típicos trucos sucios …).
Simulemos este cambio con código. El ejemplo, no es un caso real, con una jerarquía compleja, muchos métodos, etc. y por tanto tenemos que hacer un esfuerzo para visualizar que ante la duda la composición es más mantenible. Si te fijas en el main() cuando se produce el cambio la solución con composición parece más limpia y trivial

```
class Trabajador{
   String nombre;
   int edad;

   public Trabajador(String nombre, int edad) {
      this.nombre = nombre;
      this.edad = edad;
   }

   void metodo1(){
      System.out.println("metodo 1 de trabajador");
```

```
    }
}
class Empleado extends Trabajador{
    String dep;

    public Empleado(String dep, String nombre, int edad) {
        super(nombre, edad);
        this.dep = dep;
    }


     void metodo(){
        System.out.println("un meotodo de empleado");
    }
}
class Autonomo extends Trabajador{
    String actividad;

    public Autonomo(String actividad, String nombre, int edad) {
        super(nombre, edad);
        this.actividad = actividad;
    }

}

//ahora Empleado y Autonomo con versión composición
class EmpleadoC{
    String dep;
    Trabajador trabajador;

    public EmpleadoC(String dep, Trabajador trabajador) {
        this.dep = dep;
        this.trabajador = trabajador;
    }

}
class AutonomoC{
    String actividad;
     Trabajador trabajador;

    public AutonomoC(String actividad, Trabajador trabajador) {
        this.actividad = actividad;
        this.trabajador = trabajador;
    }

}
class App {
    public static void main(String[] args) {
        Empleado chuskiEmp=new Empleado("mantenimiento","CHUSKI",45);
        //AHORA RESULTA QUE CHUSKI SE HIZO AUTONOMO
        //el objeto trabajador al ir encapsulado dentro del objeto Empleado puede ser complicado
        //sacarlo para fuera, en este ejemplo es fácil pero en un caso real puede ser peliagudo
        Autonomo chuskiAutonomo=new Autonomo("Electricista",chuskiEmp.nombre,chuskiEmp.edad);

        //el cambio es más natural y sencillo con composición
        Trabajador chuski2=new Trabajador("CHUSKI",45);
        EmpleadoC chuski2Emp=new EmpleadoC("mantenimiento",chuski2);
        //ahora resulta que chuski si hizo autónomo y se cambia así de fácil
        AutonomoC chuski2Autonomo= new AutonomoC("Electricista",chuski2);


   }

}
```

no dejar de ser un ejemplo que se le pueden poner mil “peros”, sólo pretende ayudar a intuir por dónde van los tiros

Ejercicio U5_B3E_E1:

Sin plantearte si es mejor herencia o composición, simplemente escribe con composición el siguiente código

```
class Figura{
    private String color;

    Figura(String color){
        this.color=color;
    }
}

class Cuadrado extends Figura{
    private double lado;

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

    double getLado() {
        return lado;
    }
}


class Circulo extends Figura{
    private double radio;

    Circulo(double radio, String color){
        super(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("adios");
    }
}
```