# Principios Adicionales de Código Limpio en POO

**1. Principio SOLID**

* Single Responsibility Principle (SRP)
  + Según este principio “una clase debería tener **una, y solo una, razón para cambiar**”. Es esto, precisamente, “razón para cambiar”, lo que Robert C. Martin identifica como “responsabilidad”.
* Open/Closed Principle (OCP)
  + Formulado por Bertrand Meyer en 1988 en su libro “[Object Oriented Software Construction](https://sophia.javeriana.edu.co/~cbustaca/docencia/POO-2016-01/documentos/Object%20Oriented%20Software%20Construction-Meyer.pdf)”: “Deberías ser capaz de extender el comportamiento de una clase, sin modificarla”. Es decir: “Las clases que usas deberían estar **abiertas para poder extenderse y cerradas para modificarse**”
* Liskov Substitution Principle (LSP)
  + Del apellido de su creador, [Barbara Liskov](https://www.eldiario.es/sociedad/Barbara-Liskov-medidas-mujeres-ingenierias_0_780422139.html), y dice que **“las clases derivadas deben poder sustituirse por sus clases base”.**
* Interface Segregation Principle (ISP)
  + “Haz interfaces que sean específicas para un tipo de cliente”, es decir, para **una finalidad concreta**.”
* Dependency Inversion Principle (DIP)
  + “**Depende de abstracciones**, no de clases concretas”.

<https://profile.es/blog/principios-solid-desarrollo-software-calidad/#1_Principio_de_Responsabilidad_Unica>

**2. Ley de Deméter (Principio de Mínimo Conocimiento)**

* Nuestro **objeto** no debería conocer las entrañas de otros objetos con los que interactúa.
* Evitar cadenas de llamadas como: getX().getY().getZ().doSomething()
* Reduce el acoplamiento entre clases

<https://devexpert.io/ley-de-demeter/>

**3. DRY (Don't Repeat Yourself)**

* Reducir el número de veces que se repite un código, para mejorar la calidad y facilitar la entrada de otras funciones y/o realizar cambios a posterior y evitar problemas.

<https://codeyourapps.com/el-principio-dry-no-te-repitas/>

<https://platzi.com/clases/3208-programacion-basica/51967-dont-repeat-yourself-dry/>

**4. KISS (Keep It Simple, Stupid)**

* Aquí se establece que **“las cosas deben ser lo más simples posibles, tanto en cuanto a diseños como en cuanto a sistemas.”**
* Evitar la complejidad, ya que cuanto más fácil sea algo mayor garantía tendremos de alcanzar un nivel de aceptación e interacción por parte del usuario.

<https://www.iebschool.com/blog/keep-it-simple-stupid-analitica-usabilidad/>

**5. Composición sobre Herencia**

* Preferir la composición de objetos sobre la herencia cuando sea posible, nos dará mayor flexibilidad, poder hacer cambios en tiempo de ejecución, un mejor encapsulamiento, etc.
* La herencia puede crear alto acoplamiento, una herencia frágil, Jerarquías profundas que sean difíciles de mantener, etc.

<https://jugnicaragua.org/composicion-sobre-la-herencia/>

**6. Inmutabilidad**

* Crear objetos inmutables cuando sea posible para poder cambiarlo o alterarlo una vez que haya sido definido

<https://www.educaopen.com/digital-lab/metaterminos/i/inmutabilidad#:~:text=La%20inmutabilidad%20se%20refiere%20a,modificarse%20una%20vez%20estén%20creados>.