



## Curso de Buenas Prácticas para Escritura de Código

### ¡No te rindas!

Necesitas una **calificación mínima de 9.0** para aprobar.

Vuelve a intentarlo en 05 horas, 44 minutos, 42 segundos

# 8.67

Calificación

# 26 / 30

Aciertos

1. ¿A quién beneficia contar con código bien escrito?

A todos los involucrados en el proyecto



2. ¿En qué nos basamos para decir que un código es de alta calidad?

Legibilidad, mantenibilidad y testeabilidad



3. ¿Qué hace a la prolijidad del código?

Respeto de estándares



4. ¿Cuáles de estos identificadores son mnemotécnicos, específicos y precisos?

enviarClavesSecretas()



5. ¿Cuál es la principal característica del código correctamente modularizado?

Muchos bloques pequeños



6. ¿Cómo se logra código reutilizable?

Mediante el uso de funciones o procedimientos que reciben parámetros



7. ¿Cómo se determina si un código está correctamente organizado?

Sus archivos contienen elementos relacionados de forma lógica (Y sus directorios contienen archivos también relacionados de forma lógica)



8. ¿Cuál de estos no es un problema del hardcoding?

Peor rendimiento de la aplicación



9. ¿Cómo puede evitarse el hardcoding?

Mediante el uso de constantes y archivos de configuración



10. ¿Cuál es el principal problema derivado de la existencia de efectos colaterales?

Dificultad para adaptar el código a nuevos requerimientos



11. ¿Qué son los principios SOLID?

Buenas prácticas del diseño Orientado a Objetos



12. ¿Qué beneficios aporta usar los principios SOLID?

Código más reutilizable y testeable



13. ¿Qué nos enseña el Single Responsibility Principle?

A crear objetos que sepan muy bien como hacer una cosa específica



14. ¿Qué nos enseña el Open Closed Principle?

Que una clase debe poder adaptarse a nuevos escenarios sin necesidad de agregar o modificar su código



15. ¿Qué nos enseña el Liskov Substitution Principle?

Cómo debe comportarse una clase que hereda de otra



16. ¿Qué nos enseña el Interface Segregation Principle?

A crear interfaces específicas



17. ¿Qué nos enseña el Dependency Inversion Principle?

Que las dependencias de un método deben ser recibidas como parámetros



18. ¿Qué es un patrón de diseño?

Un modelo de solución a una problemática común



19. ¿Cuántas instancias de un Singleton existen en una aplicación?

1



20. ¿En qué casos conviene utilizar el patrón Factory?

Cuando la creación de instancias de una clase es especialmente compleja



21. ¿En qué casos conviene utilizar el patrón Command?

Cuando se requiera realizar una misma funcionalidad independientemente del modo en que se ejecuta la aplicación



22. ¿Para qué sirve el Testing Automatizado?

Para mejorar las chances de atrapar errores antes de que lleguen a producción



23. ¿Qué ventaja tiene el testing automatizado respecto del testing manual?

Es menos costoso



---

24. ¿Qué prueba el Unit Testing?

Unidades aisladas de software



---

25. ¿Qué prueba el Integration Testing?

El modo en que las unidades (que se suponen correctas) interactúan entre sí



---

26. ¿Qué propone el Test Driven Development?

Escribir primero las pruebas y luego el código



---

27. ¿Cuál de esos es un beneficio de usar Test Driven Development?

Se escribe únicamente el código necesario



---

28. ¿Para qué sirve un Pull Request?

Para dar a otro miembro del equipo la posibilidad de revisar el código antes de integrarlo al repositorio principal



---

29. ¿Para qué sirve documentar nuestro código?

Para facilitar la continuación del desarrollo por otro miembro del equipo o por el mismo aún si no se ha estado trabajando activamente por un tiempo



---

30. ¿Qué tan complejo es escribir código de alta calidad?

Nada complejo



---

**REGRESAR**