SOLID Principle Analysis and Application

Jeancarlo Santi

SOLID Principle	Description	Application in Current Code	Comments	Recommendations
Single Responsibility	A class should have only one reason to change, meaning it should have a single responsibility.	**No**	The class handles UI, timing management, and business logic. It should be split into separate classes.	Refactor the class into distinct components, each responsible for only one aspect of the functionality.
Open/Closed	Entities should be open for extension but closed for modification.	**No**	The current class is not designed to allow extensions without modifications.	Implement an abstract class or interface to allow future extensions without altering the base class.
Liskov Substitution	Objects of a derived class should be replaceable with objects of the base class without altering the program's functionality.	**Not Applicable** (No inheritance in the current code)	This principle is not applicable since inheritance is not used in the code.	Consider using inheritance where appropriate to promote reuse and maintainability.
Interface Segregation	Clients should not be forced to depend on interfaces they do not use.	**Not Applicable** (No interfaces in the current code)	This principle is not applicable since interfaces are not used in the code.	Introduce interfaces that are fine-tuned to the specific needs of the clients, avoiding bloated dependencies.
Dependency Inversion	Dependencies should be	**No**	The `CyberManagementPan`	Introduce an abstraction layer

	on abstractions, not concrete details. Abstractions should not depend on concrete details.		`el` class depends directly on the concrete implementation of `CyberManager`.	(interface or abstract class) to decouple the `CyberManagementPanel` from the concrete implementation.