BLOCK NONCE #2: Depecency Injection

What?

Dependency Injection (DI) is a design pattern where an object receives its dependencies from an external source rather than creating them itself. It is like giving your code a toolbox instead of making it build its own tools.

Why?

Decouples components – making it way easier to swap parts without breaking everything.
Improves testability - you can inject fake dependencies (mocks or stubs) when testing.

How?

```
// Without DI
struct WalletService {
  provider: BlockchainProvider,
}

impl WalletService {
  fn new() -> Self {
    WalletService {
      // hard-coded dependency
      provider: BlockchainProvider::connect()
    }
  }
}
```

```
// With DI
struct WalletServiceDI<'a> {
  provider: &'a dyn BlockchainProviderTrait,
}

impl<'a> WalletServiceDI<'a> {
  fn new(provider: &'a dyn BlockchainProviderTrait) -> Self {
    // The provider is passed in
    WalletServiceDI { provider }
  }
}
```

```
// DI usage

// In production
struct RealProvider;
impl BlockchainProviderTrait for RealProvider {
  fn query_balance(&self, address: &str) -> u64 {
    // Real chain logic here
  }
}

// In tests
struct MockProvider;
impl BlockchainProviderTrait for MockProvider {
  fn query_balance(&self, _address: &str) -> u64 {
    // return mock balance
  }
}
```