Cheatography

Object Oriented Rules Cheat Sheet

by [deleted] via cheatography.com/24206/cs/5557/

SOLID

Single Responsibility Principle

A class changes for only one reason

Open/Closed Principle

A class should be open for extension, closed for editing

Liskov's Substitution Principle

Derived types should cleanly and easily replace base types

Interface Segregation Principle

Favor multiple single-purpose interfaces over composite

Dependency Inversion Principle

Concrete classes depend on abstract-ions, not vice-versa

SOLID principles

Common Refactorings

Encapsulate Field

Generalize Type

Type-Checking ⇒ State/Strategy

Conditional ⇒ Polymorphism

Extract Method

Extract Class

Move/Rename Method or Field

Move to Superclass/Subclass

More Refactorings

Basic OO Terms

Abstraction

The process of separating ideas from specific instances of those ideas at work.

Polymorphism

The provision of a single interface to entities of different types. Subtyping.

Inheritance

When an object or class is based on another object or class, using the same implementation; it is a mechanism for code reuse. The relationships of objects or classes through inheritance give rise to a hierarchy.

Encapsulation

Enclosing objects in a common interface in a way that makes them interchang-eable, and guards their states from invalid changes.

Other Principles

DRY - Don't repeat yourself

Duplication should be abstracted

Hollywood Principle

"Don't call us, we'll call you"

YAGNI - You Ain't Gonna Need It

Only code what you need now

KISS - Keep it simple, stupid!

Favor clarity over cleverness

Law of Demeter

Only talk to related classes

Other Principles (cont)

Convention Over Configuration

Defaults cover 90% of uses

Encapsulation

What happens in Vegas...

Design By Contract

And then write tests

Low Coupling

Minimize the dependencies

Common Closure Principle

Classes that change together, stay together

Avoid Premature Optimization

Don't even think about optimization unless your code is working

Separation of Concerns

Different functionalities are distinctly managed

Embrace Change

Expect and welcome any changes

Basic Principles

Encapsulate what varies

Code to an interface rather than to an implementation

Each class in your application should have only one reason to change

Classes are about behavior and functiona-lity

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Design Patterns

Abstract Factory	Creational
Builder	Creational
Factory Method	Creational
Prototype	Creational
Singleton	Creational
Adapter	Structural
Bridge	Structural
Composite	Structural
Decorator	Structural
Facade	Structural
Flyweight	Structural
Proxy	Structural
Chain of Responsibility	Behavioral
Command	Behavioral
Interpreter	Behavioral
Iterator	Behavioral
Mediator	Behavioral
Memento	Behavioral
Observer	Behavioral
State	Behavioral
Strategy	Behavioral
Template Method	Behavioral
Visitor	Behavioral

Favor the following over inheritance

Delegation

> When you hand over the responsibility for a particular task to another class or method.

Composition

> Use behavior from a family of other classes, and change that behavior at runtime.

Aggregation

> When one class is used as part of another class, but still exists outside of that other class.

Access Modifiers

Private	Only inside the same class instance
Protected	Inside same or derived class instances
Public	All other classes linking/referencing the class
Internal	Only other classes in the same assembly
Protected Internal	All classes in same assembly, or derived classes in other assembly
Static	Accessible on the class itself (can combine with other accessors)

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com