|  |  |  |
| --- | --- | --- |
| **1** | Design Patterns | |
| This Cheat Sheet provides a quick reference to the original 23 **Gang of Four** (**GoF**) design patterns, as listed in the book Design Patterns: Elements of Reusable Object-Oriented Software. Each pattern includes class diagrams, explanation, usage, information, and a real world example. | | |
| |  |  | | --- | --- | |  | Creational Patterns Used to construct object such that they can be decoupled from their implementing system | |  | Structural Patterns Used to form large object structures between many disparate objects. | |  | Behavioral Patterns Used to manage algorithms, relationships, and responsibilities between objects. |  |  |  | | --- | --- | | Object Scope | Deals with object relationships that can be changed at runtime. | | Class Scope | Deals with class relationships that can be changed at compile time. | | | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | Abstract Factory |  | Decorator |  | Prototype | |  | Adapter |  | Façade |  | Proxy | |  | Bridge |  | Factory Method |  | Observer | |  | Builder |  | Flyweight |  | Singleton | |  | Chain of Responsibility |  | Iterator |  | Strategy | |  | Template Method |  | Mediator |  | Command | |  | Composite |  | Memento |  | Visitor | |
| |  |  |  |  | | --- | --- | --- | --- | | ABSTRACT FACTORY | | | Object Creational | | AZERTY | | | | | WHAT ? | azerty | | | | WHEN ? | azerty | | | | EXAMPLE | | azerty | |  |  |  |  |  | | --- | --- | --- | --- | | ABSTRACT FACTORY | | | Object Behavioral | | AZERTY | | | | | WHAT ? | azerty | | | | WHEN ? | azerty | | | | EXAMPLE | | azerty | | | | |  |  |  |  | | --- | --- | --- | --- | | ABSTRACT FACTORY | | | Object Structural | | AZERTY | | | | | WHAT ? | azerty | | | | WHEN ? | azerty | | | | EXAMPLE | | azerty | | |
| Creational Patterns | | |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  | SINGLETON | | **Object Creational** | |  | | | | | | WHAT ? | | | Only one instance of a class is allowed within a system. | | | WHEN ? | | | * Exactly one instance of a class is required. * Controlled access to single object is necessary. * There are two forms of Singleton Instantiation:   + **Early** : object creation takes place at the load time.   + **Lazy** : object creation is done according to requirement   + **Static Block Initialization**: class loading sequence.   + **Bill Pugh solution**: for java memory model changes. | | | EXAMPLE | | | Most languages provide some sort of system that allows the language to interact with the native operating system. Since the application is physically running on only one operating system there is only ever a need for a single instance of this system object. The singleton pattern would be implemented by the language runtime to ensure that only a single copy of the system object is created and to ensure only appropriate processes are allowed access to it. | | | |  |