**3.2 )**

**\* GENERICOS**

Los tipos genéricos permiten forzar la seguridad de los tipos, en tiempo de compilación, en las colecciones (u otras clases y métodos que utilicen tipos parametrizados).

**\* COLECCIONES**

Las colecciones son objetos que referencian a un grupo de objetos, las colecciones solamente mantienen referencias a objetos de tipo object.

\* **PATRON DE DISEÑO**

Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Debemos tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios).

\* **PATRON DE CONSTRUCCION**

Los patrones de construcción tienen la vocación de abstraer los mecanismos de creación de objetos. Un sistema que utilice estos patrones se vuelve independiente de la forma en que se crean los objetos, en particular, de los mecanismos de instanciación de las clases concretas.

\* **PATRON DE ESTRUCTURACION**

El objetivo de los patrones de estructuración es facilitar la independencia de la interfaz de un objeto o de un conjunto de objetos respecto de su implementación.

\* **PATRON DE COMPORTAMIENTO**

El objetivo de los patrones de comportamiento consiste en proporcionar soluciones para distribuir el procesamiento y los algoritmos entre los objetos.

\* **FAMILIA PATRON CONSTRUCCION**

**-** Abstract Factory

**-** Builder

**-** Factory Method

**-** Prototype

**-** Singleton

\* **FAMILIA PATRON DE ESTRUCTURACION**

**-** Adapter

**-** Bridge

**-** Composite

**-** Decorator

**-** Facade

**-** Flyweight

**-** Proxy

\* **FAMILIA PATRON DE COMPORTAMIENTO**

**-** Chain of responsibility

**-** Command

**-** Interpreter

**-** Iterator

**-** Mediator

**-** Memento

**-** Observer

**-** State

**-** Strategy

**-** Template Method

**-** Visitor

\* **PATRONES GOF**

**Abstract Factory**

Tiene como objetivo la creación de objetos reagrupados en familias sin tener que conocer las clases concretas destinadas a la creación de estos objetos.

**Builder**

Permite separar la construcción de objetos complejos de su implementación de modo que un cliente pueda crear estos objetos complejos con implementaciones diferentes.

**Factory Method**

Tiene como objetivo presentar un método abstracto para la creación de un objeto reportando a las subclases concretas la creación efectiva.

**Prototype**

Permite crear nuevos objetos por duplicación de objetos existentes llamados prototipos que disponen de la capacidad de clonación.

**Singleton**

Permite asegurar que de una clase concreta existe una única instancia y proporciona un método único que la devuelve.

**Adapter**

Tiene como objetivo convertir la interfaz de una clase existente en la interfaz esperada por los clientes también existentes para que puedan trabajar de forma conjunta.

**Bridge**

Tiene como objetivo separar los aspectos conceptuales de una jerarquía de clases de su implementación.

**Composite**

Proporciona un marco de diseño de una composición de objetos con una profundidad de composición variable, basando el diseño en un árbol.

**Decorator**

Permite agregar dinámicamente funcionalidades suplementarias a un objeto.

**Facade**

Tiene como objetivo reagrupar las interfaces de un conjunto de objetos en una interfaz unificada que resulte más fácil de utilizar.

**Flyweight**

Facilita la compartición de un conjunto importante de objetos con granularidad muy fina.

**Proxy**

Construye un objeto que se substituye por otro objeto y que controla su acceso.

**Chain of responsibility**

Crea una cadena de objetos tal que si un objeto de la cadena no puede responder a una petición, la pueda transmitir a sus sucesores hasta que uno de ellos responda.

**Command**

Tiene como objetivo transformar una consulta en un objeto, facilitando operaciones como la anulación, la actualización de consultas y su seguimiento.

**Interpreter**

Proporciona un marco para dar una representación mediante objetos de la gramática de un lenguaje con el objetivo de evaluar, interpretándolas, expresiones escritas en este lenguaje.

**Iterator**

Proporciona un acceso secuencial a una colección de objetos sin que los clientes se preocupen de la implementación de esta colección.

**Mediator**

Construye un objeto cuya vocación es la gestión y el control de las interacciones en el seno de un conjunto de objetos sin que estos elementos se conozcan mutuamente.

**Memento**

Salvaguarda y restaura el estado de un objeto.

**Observer**

Construye una dependencia entre un sujeto y sus observadores de modo que cada modificación del sujeto sea notificada a los observadores para que puedan actualizar su estado.

**State**

Permite a un objeto adaptar su comportamiento en función de su estado interno.

**Strategy**

Adapta el comportamiento y los algoritmos de un objeto en función de una necesidad concreta sin por ello cargar las interacciones con los clientes de este objeto.

**Template Method**

Permite reportar en las subclases ciertas etapas de una de las operaciones de un objeto, estando éstas descritas en las subclases.

**Visitor**

Construye una operación a realizar en los elementos de un conjunto de objetos. Es posible agregar nuevas operaciones sin modificar las clases de estos objetos.