# Temario

El presente documento detalla la lista de temas vistos en las clases correspondientes al TEÓRICO del curso. Para obtener la lista de temas incluidos en el parcial es necesario agregar los temas vistos en el práctico.

## Conceptos básicos del paradigma de objetos

Contenido:

- Introducción a DOO.
- Definición de clase y objeto, conceptos básicos de objetos
- Objetos - clases - UML (objeto, clase, atributos, visibilidad, métodos).
- Objeto: entidad tangible / identidad, estado y comportamiento.
- Paquetes.
- Relaciones entre clases (conceptos y modelado), implementación de relaciones
- Asociación, agregación, composición, y delegación.
- Adornos UML y conceptualmente: roles, multiplicidad, navegabilidad, notas, estereotipos.
- Clase de asociación.

Bibliografía:

Conceptos básicos de objetos: Object Oriented Analysis and Design. Grady Booch. Capítulos 1 a 5.
UML: Learning UML 2.0. Hamilton, Miles. Capítulos 1 (Introducción), 4 y 5.

## Herencia y Polimorfismo

Contenido:

- Conceptos de herencia e implementación de herencia.
- Conceptos de polimorfismo y ejercicios.
- Profundizar en el concepto de Interfaz.
- Interfaz Vs Clase Abstracta. Composición vs Herencia.
- Impacto de la herencia, la utilización de interfaces y polimorfismo en el diseño.

Bibliografía: Object Oriented Analysis and Design. Grady Booch. Capítulos 1 a 5.

# Clean Code

Contenido:

- Capítulo 1.- Clean Code, el costo asociado al mal código, ¿qué es clean Code?
- Capítulo 2.- Nombres Significativos, reglas para nombrar elementos de código: variables, funciones, clases, etc.
- Capítulo 3.- Funciones, reglas para escribir funciones relativas a tamaño, responsabilidad, nivel de abstracción, nombres, manejo de parámetros, etc.
- Capítulo 4.- Comentarios, buenos vs. malos comentarios.
- Capítulo 5.- Formato, ¿por qué es importante formatear el código?, reglas para formato vertical y horizontal.
- Capítulo 6.- Objetos y estructuras de datos, abstracción, encapsulamiento, Ley de Demeter.
- Capítulo 7.- Manejo de Errores, importancia del manejo de errores, buenas prácticas.
- Capítulo 8.- Límites, buenas prácticas para integrar código de terceros a nuestra aplicación, manejo de dependencias.
- Capítulo 10.- Clases, buenas prácticas para el diseño de clases, SRP, cohesión.
- Capítulo 9.- Pruebas Unitarias, como escribir buenos casos de prueba unitarios (con TDD).
- Capítulo 12.- Diseño emergente, reglas para obtener un buen diseño a partir de prácticas de programación (luego TDD).

Bibliografía:

Clean Code: A Handbook of Agile Software Craftsmanship. Robert C. Martin. Los capítulos mencionados en el detalle.

## TDD

Contenido:

- Metodología.
- Beneficios, limitaciones y desafíos.
- Tipos de test: Unitarios, Integración, Funcionales.
- Test unitarios: FIRST (Fast, Independent, Repeatable, Self validating, Timely)
- Ciclo Red-Green
- Pruebas antes que el código, no tener código que no tenga pruebas
- Mínima implementación para que pasen las pruebas
- Relación con refactoring
- Relación con el diseño. Idea de "Diseño emergente"

Bibliografía:

Test driven development by example. Kent Beck. Introducción.
Clean Code. Robert C. Martin. Capítulo 9

## Refactoring

Contenido:

- ¿Qué son y para que sirven las técnicas de refactoring?
- ¿Porqué son necesarias?
- Beneficios y potenciales problemas
- Situaciones que sugieren aplicar refactoreo, "code smells".
- Relación entre refactoring y testing automático.

Bibliografía:

Refactoring, Improving the design of existing code. Martin Fowler.
Capítulos 1 al 5.

## GRASP

Contenido:

- ¿Qué son y para que sirven los patrones de asignación de responsabilidades?
- Experto
- Creador
- Controlador
- Bajo Acoplamiento
- Alta Cohesión
- Polimorfismo
- Fabricación Pura
- Indirección
- Ley de Demeter

Bibliografía:

Introducción al análisis y diseño orientado a objetos. Craig Larman. Capítulo 16: GRASP.

## Principios SOLID

Contenido:

- ¿Qué son los principios fundamentales de diseño?
- Single Responsibility Principle
- Open/Closed Principle
- Liskov Substitution principle
- Interface Segregation Principle
- Dependency Inversion Principle
- En todos los casos, es necesario comprender el enunciado de cada principio, y ser capaz de identificar si se cumple o no en una situación dada, pudiendo justificar la respuesta y proponer cambios para mejorar el diseño en base a ellos.

Bibliografía:

Agile principles, patterns, and practices. Martin, Robert C.

Lectura Recomendada:

Principles of Object Oriented Design. Robert C. Martin. Disponible en http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

## Patrones de Diseño

Contenido:

- ¿Qué son los patrones de diseño y para que sirven?
- Singleton
- Facade
- Strategy
- Template Method
- Builder
- En todos los casos, es necesario comprender la intención del patrón, sus participantes, colaboraciones y consecuencias. Se espera que se pueda identificar el patrón a utilizar frente a un problema específico de diseño, aplicándolo en un contexto concreto.

Bibliografía:

Design Patterns, Elements of Reusable Object-Oriented Software. E. Gamma et al. Capítulo 1 (Introducción) y las secciones correspondientes a cada uno de los patrones antes mencionados.

## UML

Contenido:

- Diagramas de Clase
- Diagramas de Paquetes
- Diagramas de Interacción

Bibliografía:

Learning UML 2.0. Hamilton, Miles. Capítulos 1 (Introducción), 4 y 5 (Diagrama de clases), 7 y 8 (Diagramas de Interacción: secuencia y comunicación).