Facultad de Ingeniería - Universidad Nacional de Cuyo P1- PROGRAMA DE ASIGNATURA			
Asignatura:	**Programación Orientada a Objetos**		
Profesor Titular:	**César Omar Aranda**		
Carrera:	**Ingeniería en Mecatrónica**		
Año: 2019	**Semestre: 2**	**Horas Semestre: 60**	**Horas Semana: 4**

OBJETIVOS

Objetivos conceptuales:

- Comprender el paradigma de la orientación a objetos, sus características, ventajas y ámbitos de aplicación.
- Comprender básicamente el proceso de desarrollo de software orientado a objetos.
- Identificar las entidades que participan en un sistema y sus comportamientos asociados.

Objetivos procedimentales:

- Definir las características de la solución de un problema bajo el paradigma de la orientación a objetos.
- Representar sistemas usando diagramas y elementos básicos de UML
- Implementar soluciones en lenguajes orientados a objetos, a partir de transcribir un modelo representado mediante UML.

Objetivos actitudinales:

- Adquirir confianza en las propias posibilidades de comprender y resolver problemas.
- Manifestar perseverancia en las tareas a desarrollar.
- Demostrar capacidad para trabajar de manera autónoma y grupal.
- Mejorar las habilidades de investigación así como su visión crítica y autocrítica de problemas y soluciones.

CONTENIDOS

UNIDAD 1: ORIENTACION A OBJETOS

1.A. Aspectos Generales

Paradigmas de programación. Conceptos generales. Ejemplos.

Análisis de inconvenientes de los enfoques solamente procedurales en la resolución de problemas complejos, en la reutilización de código y en el mantenimiento.

Programación Orientada a Objetos vs Basada en Objetos. Ventajas y desventajas.

Panorama de lenguajes orientados a objetos. Actualidad y tendencias

IDE, CASE y otras herramientas de desarrollo. Construcción de Aplicaciones OO.

Diseño e Implementación. Beneficios y limitaciones de la generación automática de software. Roundtrip.

1.B. Paradigma Orientado a Objetos

Conceptos fundamentales de la Orientación a Objetos. Objeto y Clase. Atributos. Operaciones.

Principios y mecanismos del paradigma orientado a objetos. Abstracción. Encapsulamiento. Jerarquías de Clases. Polimorfismo. Modularidad. Instanciación. Ocultamiento. Comunicación entre objetos: Mensajes. Herencia. Tipos de Herencia.

Ejemplos y ejercitación.

UNIDAD 2: Proceso de Desarrollo y Representación de Sistemas con UML

2.A. Metodología de construcción de software OO

Proceso de Modelado Orientado a Objetos. Conceptos. Fases. Metodología. Generalidades. Ejemplos.

Principios de diseño. Patrones de Diseño. Concepto. Generalidades. Ejemplos.
Documentación de un sistema. Relación con el Manual del Usuario.

2.B. UML

Lenguaje de Modelado Unificado. Generalidades. Aplicabilidad. Elementos.
Diagramas Estructurales y de Comportamiento. Vistas.
Diagrama de Casos de Uso. Actor. Caso de Uso. Relaciones.
Diagrama de Clases. Relaciones. Asociación. Agregación. Composición.
Diagrama de Secuencia. Diagrama de Actividad. Diagrama de Estados.
Ejemplos y ejercitación.

UNIDAD 3: Programación Orientada a Objetos

3.A. Elementos fundamentales del lenguaje

Análisis comparado de similitudes y diferencias del lenguaje C++ con el lenguaje C.
Revisión de generalidades, sintaxis, semántica y elementos básicos de los lenguajes C++ y Python. Tipos de datos simples y compuestos. Operadores. Expresiones. Conversión de tipo. Declaración de variables. Constantes. Punteros y referencias. Flujos estándares de entrada/salida. Sentencias fundamentales de flujo secuencial y de control de flujo. Gestión de errores y excepciones.
Ejemplos. Ejercitación.

3.B. Implementación del paradigma de objetos en lenguaje C++

Clases. Atributos y Métodos. Definición. Instanciación. Visibilidad.
Métodos constructores y destructores.
Clase Abstracta. Herencia simple y múltiple. Reutilización. Colecciones.
Polimorfismo. Sobrecarga de métodos. Sobrecarga de operadores. Reescritura de métodos.
Ejemplos. Ejercitación.

3.C. Implementación del paradigma de objetos en lenguaje Python

Clases y objetos nativos en Python. Tipos básicos y colecciones: cadenas, listas, tuplas y diccionarios. Funciones.
Clases personalizadas. Atributos y Métodos. Definición. Instanciación.
Métodos especiales. Métodos constructores y destructores.
Herencia simple y múltiple. Reutilización. Polimorfismo.
Consideraciones particulares de OO en Python. Sobrecarga, encapsulamiento y control de acceso.
Ejemplos. Ejercitación.

3.D. Librerías especiales y programación avanzada

Librerías complementarias. Incorporación y utilización en un proyecto.
Persistencia: concepto. Serialización. Archivos. Bases de datos.
Gestión de dispositivos de entrada/salida. Conectividad TCP/IP.
Introducción al uso de: menús de usuario en consola, aplicaciones cliente/servidor, interfaces gráficas de usuario, mecanismos RPC.
Ejemplos. Ejercitación.

METODOLOGÍA DE ENSEÑANZA

Se considera que cada clase es eminentemente teórico-práctica, aplicando en forma inmediata los conceptos expresados teóricamente, por lo que el trabajo, en general, responderá al de aula-taller.

Las clases teóricas se presentan sobre pizarra o con elementos multimedia, incluyendo ejemplos de aplicación de distinto nivel de completitud y complejidad.

Se plantean trabajos prácticos, casos de estudio, investigación y/o de extensión.

Algunos de ellos tienen carácter obligatorio (se solicita y califica tanto su desarrollo como su presentación) y otros opcionales (son sólo a efectos de ejercitación de los estudiantes, de fijación de

conceptos o discusión). Según las características del alumnado y del ciclo lectivo, los trabajos se distribuyen para ser llevados adelante de manera individual y/o grupal.

El seguimiento de estudiantes se realiza registrando asistencia, cumplimiento y participación.

En principio el desarrollo de cada clase cuenta con tres momentos sucesivos:

- La introducción: Donde se expone teóricamente un tema nuevo o se realiza el análisis del código de un ejemplo simple ya resuelto.
- La elaboración: Donde se construye la solución a un problema acorde al tema explicado anteriormente y que permite se pongan de manifiesto las principales dudas conceptuales y prácticas.
- El cierre: Donde se analizan críticamente diferentes soluciones obtenidas y/o se incluyen conclusiones pertinentes al tema.

Muchos de los conceptos de UML, uso del IDE y principios de diseño y programación, son desarrollados durante el cursado insertos entre otros contenidos, de manera no secuencial (según se detallan en el programa de contenidos). En las unidades 1 y 2 se plantean conceptos generales, que son trabajados de manera específica y con mayor extensión durante el desarrollo de la unidad 3.

Uso de mecanismos compartidos sobre redes y comunicaciones para la organización/distribución de los recursos de estudio así como para la realización de diversas actividades docentes complementarias a las propuestas de manera presencial.

En caso de que quedase algún tema sin dictar, se provee la bibliografía adecuada.

Actividad	Carga horaria por semestre
Teoría y resolución de ejercicios simples	20
Formación práctica	
Formación Experimental – Laboratorio	20
Formación Experimental - Trabajo de campo	0
Resolución de problemas de ingeniería	10
Proyecto y diseño	10
Total	**60**

BIBLIOGRAFÍA

Bibliografía básica

Autor	Título	Editorial	Año	Ejemp. en biblioteca
DEITEL H. M. Y DEITEL P. J.	Cómo Programar en C/C++ 6ª ed.	Prentice-Hall	2009	0
STROUSTRUP, B.	Programming. Principles and Practice Using C++	Pearson Education	2009	0
BRONSON, Gary	C++ para Ingeniería y Ciencias, 2da edición	Cengage Learning	2007	0
VON ROSSUM, G.	El Tutorial de Python (v3)	Python Software Foundation	2017	(1)
PEREZ CASTAÑO, A.	Python Fácil	Marcombo	2016	0
BEAZLEY, D. y JONES, B.	Python Cookbook	O'Reilly Media. 3º ed.	2013	0

Bibliografía complementaria

Autor	Título	Editorial	Año	Ejemp. en biblioteca
F. XHAFA, P. PAU VAZQUEZ, A. JORDI y otros	Programación en C++ para ingenieros	Thomson-Paraninfo	2006	0
CEBALLOS SIERRA, F. J.	C/C++ Curso de Programación, 3ª ed.	Ra-Ma	2007	0
ECKEL, Bruce	Thinking In C++, Volumen 2	Mindview, Inc	2004	0

ELLIS, M.A. ; STROUSTRUP, B.	C++ Manual De Referencia Con Anotaciones	Díaz De Santos	2004	0
CEBALLOS SIERRA, F. J.	Enciclopedia del lenguaje C++	AlfaOmega	2004	2
ACERA GARCIA, M. A.	C/C++ (ed. revisada y actualizada 2012)	Anaya Multimedia	2012	0
ECKEL, Bruce	Aplique C++	McGraw Hill	1991	2
VON ROSSUM, G.	El Tutorial de Python (v2)	Python Software Foundation	2009	(1)
GONZÁLEZ DUQUE, R.	Python para todos	(2)	2008	(3)

(1) Disponible en http://tutorial.python.org.ar/
(2) Distribuido bajo una licencia Creative Commons Reconocimiento 2.5 España
(3) Disponible en http://mundogeek.net/tutorial-python/

EVALUACIONES

Durante el período de clases, se prevé un sistema de evaluación continua consistente en un registro de los trabajos individuales y grupales realizados en clase.

Esto no implica la entrega de todos los trabajos al profesor por parte del estudiante para su revisión. En el caso de la elaboración de programas propuestos, es el estudiante quien debe lograr la habilidad de obtener software funcional y verificar la correctitud de su solución a partir del producto final obtenido. Mostrando finalmente sus resultados o conclusiones.

En base a las características del grupo y tiempos disponibles se agregará la elaboración de trabajos monográficos para su entrega y/o exposición.

Se prevé realizar 2 (dos) evaluaciones teórico-prácticas y 1 (una) evaluación práctica.

Las primeras de ellas consisten en cuestionarios, con una única instancia de recuperación, abordando contenidos conceptuales y de aplicación práctica que corresponden a un período indicado en clase oportunamente.

La última corresponde a un trabajo especial de programación, que dada la modalidad de su desarrollo (presenta varias instancias de revisión), no es recuperable. Este Trabajo Práctico será resuelto y presentado al resto del curso de manera individual o grupal, dependiendo de la población del curso.

Se proponen además una serie de actividades prácticas de desarrollo individual obligatorio para seguimiento y control.

Todas las evaluaciones deben dejar una constancia documental, ya sea en formato de papel o digital. Esta evaluación (o una copia) es devuelta al estudiante con las correcciones pertinentes, eventualmente acompañadas de observaciones verbales individuales y/o grupales.

El criterio de evaluación a seguir, es adelantado (informado) en clase.

Para obtener la regularidad o la promoción directa en la asignatura se tienen en cuenta las calificaciones obtenidas en la evaluación parcial, en la aprobación del Trabajo Práctico Especial y en el cumplimiento de los trabajos prácticos de seguimiento obligatorios.

Para aprobar la asignatura durante el cursado, esto es alcanzar la Promoción, se deben satisfacer las mismas condiciones de la regularidad y haber obtenido una calificación mínima de 70% en cada evaluación (tanto en los cuestionarios como en el Trabajo Práctico Especial) así como un promedio mínimo de 7 (considerando las 3 instancias de evaluación).

Para la aprobación de la asignatura, en condición de Estudiante Regular, se propone la presentación y defensa individual de un producto software elaborado individualmente (puede ser la continuación del Trabajo Especial realizado durante el cursado).

Éste producto consiste en la implementación de una aplicación que debe ejecutar libre de fallas (jamás terminar anormalmente), aplicando la totalidad de los conceptos fundamentales aprendidos en clase y en la confección de un informe técnico de la aplicación que incluye elementos conceptuales, elementos de diseño y un manual de usuario.

El “objetivo” del software y los temas abordados pueden ser propuestos por el estudiante y acordados con el profesor con una prudente anticipación a la fecha elegida para el examen final, tanto para permitir la corrección como la aprobación previa del producto resultante.

La defensa consiste en una exposición coloquial sobre los conceptos fundamentales y estrategias utilizadas en la confección del producto. La finalidad de esta exposición no es otra que la de corroborar autoría.

Para la aprobación de este espacio curricular en condición de Alumno Libre, se debe elaborar, presentar y aprobar un Trabajo Especial similar al indicado para el estudiante en condición Regular.

Sin embargo, para acceder a la presentación y coloquio correspondientes, y dado que la asignatura está orientada a desarrollar habilidades de programación bajo un paradigma específico el

estudiante debe demostrar las mismas, resolviendo previamente una consigna de programación de desarrollo individual sobre un producto software acotado, asignado por el profesor en el momento del examen y resuelto en un tiempo limitado a un máximo de 1 hora y media.

El caso provisto, consiste en un breve texto descriptivo del problema a resolver acompañado de una o más representaciones UML. Implica tanto el uso de clases propias del lenguaje como de clases definidas por el programador, y precisa aplicar algunos de los conceptos fundamentales de este programa de estudio (como mecanismos de herencia y polimorfismo, almacenamiento de información usando clases de colección, persistencia, sobrecarga de métodos, sobrecarga de operadores, plantillas, u otros) según la consigna.

El problema se diseña e implementa en el momento del examen y no requiere de un coloquio de defensa.

Más allá que el programa deba compilar y ejecutar sin errores, es fundamental para su aprobación demostrar que se han aplicado de manera correcta los conceptos y mecanismos tanto de la OO como de el/los lenguajes indicados (Python/C++). Una vez comprobado esto último, la calificación se asigna analizando la implementación realizada.

Con respecto a otros detalles relacionados con adquirir la regularidad y/o la aprobación de la asignatura se siguen los lineamientos generales fijados para la carrera, tanto de tipo académico como administrativos.

Programa de examen

No Aplicable