FCEFyN
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS y NATURALES

UNC Universidad Nacional de Córdoba

Asignatura: **Programación Avanzada**

Código:10-09803	RTF	7
Semestre: Tercero	Carga Horaria	96
Bloque: Tecnologías Básicas	Horas de Práctica	48

Departamento: Computación

Correlativas:

- Algoritmos y Estructuras de Datos

Contenido Sintético:

- Visión de conjunto e Historia
- Diseño y Programación Orientada a Objetos
- Prueba, verificación y validación de software
- Uso de interfaces de programación de aplicaciones
- Patrones de diseño de software
- Diseño de aplicaciones. Visualización de datos.
- Experiencia de Usuario

Competencias Genéricas:

- CG1: Identificar, formular y resolver problemas de ingeniería.(B)
- CG2: Concebir, diseñar y desarrollar proyectos de ingeniería (sistemas, componentes, productos o procesos).(B)
- CG9: Aprender en forma continua y autónoma.(B)

Aprobado por HCD: 1042-HCD-2023 RES: Fecha:27/11/2023

Competencias Específicas:

CE1.1 Analizar, especificar, diseñar, proyectar y desarrollar programas de computadoras en lenguajes de alto y bajo nivel.
CE1.2 Analizar, especificar, diseñar y proyectar arquitectura de sistemas informáticos.
CE1.3 Conocer, desarrollar nuevos e implementar algoritmos y estructuras de datos.

CE4.3 Analizar, diseñar, programar, implementar, probar, depurar y evaluar hardware y software para sistemas de computación de propósitos específicos.
CE4.10 Analizar, interpretar, modelar, diseñar interfaces humano-máquina en sistemas de software y software-hardware optimizando la experiencia de usuario.

Presentación

La Programación Orientada a Objetos se inicia con el sistema de simulación denominado Simula en los años 60 y posteriormente recibe un gran impulso teórico de sus fundamentos con el grupo de investigación del Xerox Parc Place y su desarrollo del lenguaje Smalltalk en los 70. La pureza del paradigma implementado en Smalltalk se ha convertido en su punto débil ya que se debe abandonar completamente el paradigma procedural y también la sintaxis de los lenguajes descendientes del Algol como Pascal y particularmente la familia del C, constituida por el C mismo, C++, C# y el Java. Un caso particular es el de Eiffel, más cercano a las ideas del Pascal y Algol, pero implementa el paradigma de objetos en forma pura, lo que lo lleva ser de difícil distribución masiva, no obstante, muchas de sus características han sido implementadas actualmente en Java y C#.

Otro aspecto que ha colaborado en la difusión del paradigma orientado a objetos han sido los Sistemas Operativos con interfaces gráficas de usuario en las cuales la metáfora de los objetos y los eventos permite elaborar una interacción con el usuario en forma natural y flexible.

Simultáneamente con el crecimiento de la Programación Orientada a Objetos se da el Análisis y Diseño Orientados a Objetos hasta desembocar en el Proceso Unificado de Desarrollo y el Lenguaje de Modelado (UML) que permite actualmente disponer de una Metodología que guíe el proceso de diseño e implementación. Al igual que en los proyectos de ingeniería, se ha descubierto que existen patrones de diseño a partir de los cuales se puede construir el software sin necesidad de tener que reinventarlo y, además, se ha consolidado la construcción de software a partir de componentes reusables durante los 90 y principios de este siglo. Actualmente, el desarrollo profesional de software se somete a estos principios que denominamos Ingeniería de Software.

En cuanto a la pedagogía de la asignatura, se ha seguido el concepto de introducir los Objetos Primero y para ello el estudiante deberá acompañar el estudio teórico del diseño orientado a objetos con la práctica de la elaboración de partes de un proyecto integral que justifique plenamente por su complejidad el paradigma de análisis, diseño e implementación adoptados. También se pone énfasis en aceptar que el desarrollo industrial de software requiere metodologías de Ingeniería de Software y de herramientas de desarrollo que amplifiquen la eficiencia y la productividad. Esto no significa que se abordará en forma completa ningún lenguaje orientado a objetos en particular ya que se enfocará en los criterios de diseño.

Contenidos

Introducción a los lenguajes de programación

Historia de los lenguajes de programación y los distintos paradigmas. Comparación entre intérpretes y compiladores; fases de la traducción de lenguajes; aspectos independientes y dependientes de la máquina. El concepto de máquina virtual, jerarquía de máquinas virtuales, lenguajes intermediarios

Objetos y Clases
Definición de clases y de objetos. Llamado de métodos y sus parámetros. Tipos de datos. Instancia de objetos y estado. Interacción básica entre objetos. Objetos como parámetros de métodos. Declaración y definición de clases: Campos, constructores y métodos.

Interacción entre objetos
Abstracción y modularización de software. Diagramas de clases y de objetos. Tipos primitivos y tipos de objetos. Creación de objetos y constructores múltiples. Llamado interno y externo de métodos. La autorreferencia a un objeto.

Agrupamiento de objetos
Agrupamiento de objetos en colecciones de tamaño flexible. Elementos básicos de una biblioteca de clases. Clases genéricas. Enumeración y gestión de colecciones. Iteradores. Colecciones fijas de objetos. Información y ocultamiento de campos y métodos. Variables de clase y constantes. Documentación de bibliotecas de clases estándar. Interfaces o implementación de métodos. Lectura y redacción de documentación de clases parametrizadas. Programación defensiva. Manejo de excepciones. Reporte de errores.

Procesamiento funcional de colecciones
Un primer vistazo a las funciones lambda. El método de colecciones forEach. Streams.

Prueba, depuración y mantenimiento de software
Pruebas de unidad. Inspectores. Pruebas positivas y negativas. Automatización de las pruebas. Pruebas de regresión. Escenarios y registro de pruebas. Uso de depuradores. Uso de aserciones.

Diseño de clases
Introducción al acoplamiento y la cohesión. Duplicación y extensión de código. Uso de encapsulado. Diseño basado en responsabilidades. Refactorización. Guías de diseño.

Herencia de clases
Jerarquías de herencia de campos y métodos de clases. Derechos de acceso a la herencia. Inicialización de instancias con herencia. Superclase y tipo de dato. Subtipos y subclases. Variables polimórficas. Conversión de tipo (Casting). Tipos estáticos y dinámicos. Sobrecarga de métodos. Búsqueda dinámica de métodos. Llamado a la superclase en los métodos. Polimorfismo de métodos. Acceso protegido. Clases abstractas. Métodos abstractos. Interfaces. Interfaces como tipos. Interfaces como especificaciones.

Construcción de interfaces gráficas de usuario

Construcción de Interfaces Gráficas de Usuario. Componentes y despliegue de la interfaz. Manejo de eventos. Bibliotecas de manejo abstracto de ventanas e Interfaces Gráficas de Usuario.

Diseño de aplicaciones
Análisis y diseño. El método de verbos/sustantivos. Descubrimiento de clases. Escenarios. Colaboración entre clases. Diseño de interfaz de clases y usuarios. Documentación. Cooperación. Modelos de desarrollo de software. Patrones básicos de diseño.

Metodología de enseñanza

Cada nuevo tema se abordará mediante clases expositivas y exposición dialogada, a fin de introducir a los estudiantes en la temática.

Los estudiantes deberán afrontar diversas actividades prácticas con los saberes conceptuales y procedimentales adquiridos previamente. El docente seguirá el proceso y orientará al estudiante en la ejecución de las actividades de práctica y en los trabajos prácticos.

Todas las actividades y material utilizado se encontrará disponible en el Aula Virtual de la Asignatura.

Evaluación

Los estudiantes deberán demostrar los conocimientos y habilidades adquiridas en tres tipos de instancias de evaluación:

Evaluación conceptual (EC): Consiste en una serie de evaluaciones breves de ejercicios que permitan demostrar comprensión de temas puntuales desarrollados en las clases anteriores. La evaluación conceptual consiste en ejercicios de programación que serán evaluados objetivamente mediante pruebas de software basadas en test unitarios, verificando el cumplimiento de los requerimientos, de forma automática a través del Aula Virtual. Para aprobar el conjunto de todas las Evaluaciones Conceptuales, se deberá alcanzar un rendimiento igual o superior al 60% del puntaje total. Se prevé un recuperatorio del 30% del puntaje total que se considerará como puntaje adicional y se suma a los ya obtenidos, no pudiendo pasar la suma del 100% del puntaje. Tienen carácter acreditativo.

Trabajo Práctico (TP): Consiste en una serie de actividades de desarrollo de la solución algorítmica a problemas propuestos por la Cátedra. Las soluciones deberán estar libres de errores sintácticos que impidan su compilación y generación de código ejecutable. La solución propuesta por el estudiante se evaluará objetivamente mediante pruebas de software basadas en test unitarios, verificando el cumplimiento de los requerimientos, de forma automática a través

del Aula Virtual. Para aprobar cada trabajo práctico, se deberá alcanzar individualmente un rendimiento igual o superior al 60%. Estas actividades son de carácter extra-áulico, para las cuales los alumnos dispondrán de no menos de 72 horas para su resolución y envío. Tienen carácter acreditativo. Si bien cada trabajo puede favorecer el desarrollo de una determinada competencia en particular y es de esperar la evidencia de esto hacia la conclusión de dicha actividad, la evaluación será contínua a lo largo de todas las actividades propuestas y se haŕá mediante el uso de rúbricas.

Examen de Promoción (EP): en la última clase, se realizará una ejercitación consistente en el desarrollo de un programa correspondiente al enunciado de un algoritmo. La implementación deberá estar libre de errores sintácticos que impidan su ejecución por parte del compilador, como requisito para su evaluación. La solución propuesta por el estudiante se evaluará objetivamente mediante pruebas de software basadas en test unitarios, verificando el cumplimiento de los requerimientos, de forma automática a través del LEV. Para ser aprobado, se deberá alcanzar un rendimiento igual o superior al 60%. Tiene carácter acreditativo. Al final del semestre cada estudiante debe haber demostrado un nivel de desarrollo mínimo de las competencias propuestas a través de los resultados de aprendizaje propuestos.
El diseño del examen permitirá evaluar si se alcanzaron los resultados de aprendizaje abajo indicados.

Condiciones de aprobación

Condiciones de regularización
El estudiante alcanzará la condición de regular al cumplir las siguientes condiciones:

1. Asistir al 80% de las clases. La asistencia será registrada en el aula virtual mediante la presentación del 60% de las EC y del 60% de los TP.
2. Alcanzar un rendimiento global igual o superior al 60% de los puntos en las EC. La presentación a la EC confiere asistencia a clase.
3. Aprobar el 60% de los TP propuestos.
4. Alcanzar el nivel de desarrollo mínimo en los resultados de aprendizaje propuestos.

Condiciones de promoción
El estudiante alcanzará la condición de promocionado al cumplir las siguientes condiciones:

1. Alcanzar la condición de alumno regular para lo cual se deben cumplir las condiciones indicadas al respecto.
2. Aprobar la actividad EP.

Cumplidas estas condiciones, la nota final de promoción se calculará según la siguiente fórmula:

Nota Final = redondear((0,3 * EC + 0,7 * EP)/10)

Examen final
Los estudiantes regulares rendirán un examen equivalente a la actividad Examen de Promoción (EP), la cual será calificada del mismo modo que para los alumnos promocionados, considerando para la variable EC el rendimiento alcanzado durante la cursada, y para la variable EP el rendimiento alcanzado en el examen final.

Los estudiantes libres rendirán un examen que constará de dos partes:

1. Una prueba de competencias con la misma metodología y objetivos que la Evaluación Conceptual (EC) que serán evaluados automáticamente en el aula virtual. La aprobación de esta primera parte es requisito excluyente para la prosecución del examen, y se deberá obtener un rendimiento igual o superior al 60%.
2. Un examen equivalente al Examen de Promoción (EP), con la misma modalidad y objetivos que el requerido a los alumnos regulares.

La Nota Final final de examen para los casos 1 y 2 se obtendrá por la siguiente expresión:

Nota Final = redondear ((0,3 * EC + 0,7 * EP)/10)

Actividades prácticas y de laboratorio

Las clases serán de carácter teórico-práctico en laboratorio de computación, en las cuales se presentarán los temas teóricos, se dará lugar al diálogo e intercambio de ideas y se resolverán actividades específicas a la temática abordada cada clase.

Resultados de aprendizaje

- Identificar problemas en el planteo de diseño y desarrollo de software.
- Formular soluciones efectivas y aplicar métodos de resolución de problemas
- Diseñar software aplicando los principios de modular, cohesión y acoplamiento adecuados
- Resolver problemas de programación aplicando nuevas herramientas y paradigmas de programación al problema más allá de los abordados en clase
- Implementar el concepto de tipo de dato y tipo abstracto de dato e identificar las características principales de un sistema de tipos.
- Explicar los fundamentos de la orientación a objetos y ser capaz de identificar las diferencias entre la representación basada en objetos y los modelos de flujo de datos.

- Aplicar el diseño modular y los conceptos de cohesión y acoplamiento.
- Desarrollar sistemas aplicando técnicas y metodologías de desarrollo: especificación de requisitos, análisis, diseño, prueba y depuración de aplicaciones orientadas a objetos.
- Utilizar un entorno de desarrollo para aplicar los fundamentos del paradigma orientado a objetos mediante un lenguaje de programación orientado a objetos..
- Diseñar software empleando lenguajes de modelado.
- Desarrollar aplicaciones informáticas utilizando diferentes estructuras de datos, aplicando algoritmos de ordenación y búsqueda sobre ellas.
- Utilizar patrones estándar de diseño para el desarrollo de aplicaciones.
- Implementar soluciones algorítmicas a problemas y ser capaz de representarlas como un software orientado a objetos.

Bibliografía

- David J. Barnes y Michael Kölling (2017), "Programación orientada a objetos con Java usando BlueJ" (Sexta edición), Pearson https://efn.biblio.unc.edu.ar/cgi-bin/koha/opac-detail.pl?biblionumber=17531
- Robert C. Martin (2012), "Código limpio: manual de estilo para el desarrollo ágil de software", Anaya https://efn.biblio.unc.edu.ar/cgi-bin/koha/opac-detail.pl?biblionumber=17536
- Bertrand Meyer (2009), "Touch of class : learning to program well with objects and contracts", Springer https://efn.biblio.unc.edu.ar/cgi-bin/koha/opac-detail.pl?biblionumber=9937