**CARTA DESCRIPTIVA (FORMATO MODELO EDUCATIVO UACJ VISIÓN 2020)**

## I. Identificadores de la asignatura

| | | | |
|---|---|---|---|
| **Instituto:** | IIT | **Modalidad:** | Presencial |
| **Departamento:** | Eléctrica y Computación | **Créditos:** | 8 |
| **Materia:** | Programación II | | |
| **Programa:** | Ingeniería en Sistemas Computacionales | **Carácter:** | Obligatoria |
| **Clave:** | IEC981100 | **Tipo:** | Curso |
| **Nivel:** | Intermedio | | |
| **Horas:** | 64 Totales **Teoría:** 50% | | **Práctica:** 50% |

## II. Ubicación

| Antecedentes: | Clave |
|---|---|
| Programación I | IEC981000 |

Consecuente:

## III. Antecedentes

Conocimientos: Fundamentos de gestión educativa y administración, elementos básicos de investigación cualitativa y cuantitativa, diseño curricular y evaluación educativa.

Habilidades: Capacidad de análisis y síntesis. Habilidades para el diseño de algoritmos usando algún lenguaje de modelado. Habilidades de lectura y discriminación de la información. Habilidades para la búsqueda, análisis y organización de información en medios electrónicos (Internet) como bases de datos, revistas, libros). Comprensión de texto técnico escrito en idioma inglés. Argumentación mediante lenguaje oral y trabajo en equipo.

Actitudes y valores: Honestidad académica, autocrítica, responsabilidad, respeto y disposición para el aprendizaje, auto motivación, aprendizaje regulado, trabajo colaborativo, personalidad

emprendedora, juicio crítico.

## IV. Propósitos Generales

Los propósitos fundamentales del curso son:

- Identificar y resolver problemas usando el paradigma de Orientación a Objetos; aplicando una metodología específica que lo lleve a seguir las fases de análisis, diseño, desarrollo, pruebas y validación.
- Dominar conceptual y prácticamente los elementos del paradigma de programación orientada a objetos, usando un lenguaje de propósito general.
- Analizar problemas y proponer soluciones algorítmicas.
- Solucionar problemas aplicando una metodología de programación orientada a objetos utilizando como herramienta un lenguaje de programación de propósito general (lenguaje C++).
- Colaborar en un equipo de desarrollo de software en el que se implementen los roles de analista, diseñador, desarrollador y *tester;* experimentando cada uno de los roles durante el semestre.
- Generar aplicaciones de software que cumplan con los aspectos de calidad marcados en la métrica de evaluación de la calidad de software.

## V. Compromisos formativos

Intelectual: El estudiante identificará y analizará problemas para ser resueltos mediante el diseño de un programa usando la metodología de orientación a objetos. Identificará los objetos y disenará las clases correspondientes que den solución a un problema determinado.

Humano: El estudiante reflexionará acerca de las implicaciones éticas de realizar programas de calidad que ayuden a resolver problemas reales.

Social: El estudiante analizará las repercusiones de realizar software de calidad y eficiente.

Profesional: El estudiante incorporará a su formación los elementos fundamentales de la programación orientada a objetos de forma que pueda diseñar, orientar, asesorar y/o animar a realizar proyectos de software, así como intervenir en la toma de decisiones para el mejoramiento de software de su institución o empresa.

## VI. Condiciones de operación

| | | | |
|---|---|---|---|
| Espacio: | Aula tradicional | | |
| Laboratorio: | Cómputo | Mobiliario: | Mesa y sillas |
| Población: | 25 - 30 | | |

Material de uso frecuente:

| | |
|---|---|
| | A) Proyector<br>C) Cañón y computadora portátil |
| Condiciones especiales: compilador de lenguaje C++ | No aplica |

**VII. Contenidos y tiempos estimados**

| Temas | Contenido | Actividades |
|---|---|---|
| **Tema 1: Breve introducción al paradigma POO**<br><br>**Número de sesiones: 7 sesiones = 16 hrs.** | 1.1 Tecnologia Orientda a Objetos<br>1.1.1 Una perspectiva histórica<br>1.1.2 Ventajas de un lenguaje orientado a objetos<br>1.2 El modelo orientado a objetos<br>1.2.1 Objetos<br>1.2.2 Clases<br>1.2.3 Herencia<br>1.2.4 Envío de mensajes<br>1.3 Características asociadas a la POO<br>1.3.1 Abstracción<br>1.3.2 Encapsulamiento<br>1.3.3 Ocultamiento<br>1.4 Lenguajes de programación orientada a objetos | **Encuadre del curso**: Descripción por parte del docente sobre la metodología de enseñanza-aprendizaje a utilizar durante el curso, así como los criterios de evaluación.<br><br>1.1 El docente explicará la tecnología orientada a objetos.<br>- El alumno escribirá un ensayo sobre la importancia de la programación orientada a objetos en base a las ventajas y desventajas del paradigma, mencionando ejemplos de aplicación, identificando a su vez la importancia que la POO tiene dentro de las necesidades de la industria, la complejidad de los sistemas y la convergencia de la tecnología.<br>1.2 El docente explicará el modelo orientado a objetos, definiendo los conceptos de: objeto, clase, herencia, mensajes, etc.<br>- Mediante el análisis de un escenario real (cocina, consultorio, aula, taller, etc.), el alumno deberá reconocer los objetos y los posibles mensajes entre ellos.<br>- El alumno definirá clases identificando atributos y comportamiento (UML).<br>- El alumno diseñará relaciones |

| | | |
|---|---|---|
| | | de herencia entre clases y sub-clases (UML).<br>1.3 El docente explicará los conceptos de abstracción, encapsulamiento y ocultamiento.<br>- El alumno investigará y comprobará el significado de los términos de abstracción, encapsulamiento y ocultamiento, además del rol que ocupan en la POO.<br>- El alumno Instalará el compilador del lenguaje de programación que se usará durante el curso (C++).<br>- El docente codificará el primer proyecto clásico de "Hola Mundo".<br>- El instructor mostrará el funcionamiento del compilador y las funciones para depurar un programa. |
| **Tema 2: Clases y objetos en POO**<br><br>**Número de sesiones: 10 sesiones = 16 hrs.** | 2.1 Representación de la información por medio de objetos.<br>2.2 Atributos o estado.<br>2.3 Métodos o comportamiento.<br>2.4 Abstracción de objetos en clases.<br>2.5 Necesidad y relevancia de los constructores de clase: por defecto y propios.<br>2.6 Métodos de acceso y modificación del estado de un objeto.<br>2.7 Modificadores de acceso: relevancia y necesidad de los modificadores público y privado.<br>2.8 Encapsulación de la información.<br>2.9 Prácticas definidas por el instructor, relativas a la unidad temática. | 2.1 -2.2 El instructor explica los conceptos de método y comportamiento.<br>2.3 El alumno define los métodos relacionados con cada tipo de objeto especificado.<br>2.4 El instructor explica el concepto de clase y cómo especificarlo mediante un diagrama (UML).<br>- El alumno define los diagramas de las clases, agregando atributos y métodos para cada uno de los objetos definidos.<br>- El instructor utiliza el diagrama de clases obtenido y lo traduce al código del lenguaje utilizado (C++).<br>2.5 El instructor explica el por qué del uso del constructor y muestra un ejemplo de programación, ilustrando los diferentes tipos de constructores.<br>- El alumno implementa los constructores necesarios para las clases diseñadas en las actividades del punto 2.4.<br><br>2.6 El instructor explica cómo |

| | | |
|---|---|---|
| | | los métodos pueden cambiar el estado de un objeto mediante el acceso directo a los atributos del mismo.<br>- El instructor muestra un ejemplo ilustrativo de modificación de atributos mediante el uso de métodos.<br>- El alumno diseña e implementa en código, los métodos necesarios para las clases especificadas (ejemplo: escenario propuesto por el instructor).<br>- El instructor codifica un programa que manipule objetos de las clases diseñadas. Lo anterior con el objetivo de verificar el comportamiento de los objetos.<br><br>2.7 El instructor explica el uso de los modificadores de acceso y puntualizará las diferencias entre las etiquetas *public* y *private*.<br>- El instructor muestra un ejemplo de uso de los especificadores de acceso.<br>- El alumno deberá posicionar los especificadores de acceso en cada una de las clases diseñadas (ejemplo: escenario propuesto por el instructor).<br>2.8 El instructor explica el concepto de encapsulación.<br>- El estudiante deberá definir, en sus propios términos, el concepto de encapsulación de datos y la importancia en la programación orientada a objetos.<br>- El instructor revisa el diagrama de clases y el código correspondiente al escenario dado como ejemplo desde el principio de la unidad temática.<br>2.9 El instructor revisa las prácticas de la unidad temática. Cada práctica debe ser entregada con su respectivo diagrama de clases así como el código correspondiente. La revisión será en el lugar de trabajo mostrando la funcionalidad del programa. |

| | | |
|---|---|---|
| **Tema 3: Sobrecarga de operadores**<br><br>**Número de sesiones: 7 sesiones = 16 hrs.** | 3.1 Sobrecarga de operadores binarios.<br>• operadores aritméticos<br>• operadores de relación<br>• operadores unarios<br>• operador de indexación<br>• operador de función<br>• operadores *new* y *delete*<br><br>3.2 Funciones amigas<br>- sobrecarga de los operadores de inserción y extracción de flujo (<<, >>).<br>3.3 Prácticas definidas por el instructor, relativas a la unidad temática. | 3.1 El instructor explicará la sobrecarga de operadores y mostrará un ejemplo en cada caso.<br>- El alumno deberá conocer la importancia de la sobrecarga y para qué sirve. Para ello escribirá una cuart illa explicando la utilidad de la sobrecarga de operadores.<br>3.2 El instructor explicará el concepto de las funciones amigas y su aplicación.<br>- El instructor define un escenario, por ejemplo: A un programador su jefe le entrega los diagramas de una aplicación que es una calculadora que tiene básicamente cuatro operaciones (suma,resta multiplicación y división), la aplicación deberá estar terminada para el día siguiente sin excusa ni pretexto. Nota: el escenario puede cambiar de acuerdo al instructor.<br>- El alumno analizará el escenario presentado para encontrar los conceptos fundamentales que finalmente se convertirán en los objetos principales que utilizará la aplicación.<br>- El alumno deberá de entregar el diagrama de clases y la codificación correspondiente.<br>3.3 El instructor revisa la prácticas de la unidad temática. Cada práctica debe ser entregada con su respectivo diagrama de clases así como el código correspondiente. La revisión será en el lugar de trabajo mostrando la funcionalidad del programa. |

| | | |
|---|---|---|
| **Tema 4: Herencia y relaciones entre clases**<br><br>**Número de sesiones: 8 sesiones = 16 hrs.** | 4.1 Comunicaciones entre distintas clases.<br>4.2 Clases que contienen objetos como atributos:<br>- Relaciones de agregación<br>- Relaciones de composición<br>- Relaciones de asociación<br>4.3 Relaciones de especialización/generalización.<br>4.4 Definición de la relación de herencia entre clases.<br>- Representación de relaciones de herencia en UML.<br>- Declaración y definición de relaciones de herencia en C++.<br>4.5 Redefinición de métodos en clases heredadas.<br>4.6 Modificador de acceso 'protegido" y posibilidades de uso.<br>4.7 Prácticas definidas por el instructor, relativas a la unidad temática. | 4.1 El instructor explica las diferentes formas de comunicación entre clases mediante el uso de un ejemplo práctico en UML.<br>- El estudiante explicará las diferentes formas de comunicación entre clases, dando ejemplos.<br>4.2 El instructor explica cada uno de los conceptos de agregación, composición y asociación usando diagramas UML.<br>- El instructor define un escenario, por ejemplo: Diseñar un pequeño programa que pueda detectar discos compactos, y debe contar con:<br>Registro de prestamos<br>Consultas por: persona, categoría, fecha de préstamo<br>- El estudiante deber de Implementar los diagramas pertinentes ( de clases, agregación, composición y asociación).<br>Nota: el escenario puede cambiar de acuerdo al instructor.<br>- El estudiante deberá implementar los diagramas de clases, relaciones de agregación, composición y asociación, usando el lenguaje C++, del escenario analizado.<br>4.3 El instructor explicará las relaciones de especialización/generalización mediante un ejemplo práctico.<br>4.4 El instructor muestra las ventajas de uso de relaciones de herencia: reutilización de código.<br>- El instructor muestra la representación de relaciones de herencia usando un ejemplo en UML.<br>- El instructor muestra la traducción del diagrama de relación de herencia a código en lenguaje C++.<br>4.5 El instructor explicará la redefinición de métodos en clases heredadas.<br>4.6 El instructor explicará el |

| | | |
|---|---|---|
| | | uso del modificador de acceso protegido en clases, mediante un ejemplo práctico.<br>4.7 El instructor revisa las prácticas de la unidad temática. Cada práctica debe ser entregada con su respectivo diagrama de clases así como el código correspondiente. La revisión será en el lugar de trabajo mostrando la funcionalidad del programa. |

**VIII. Metodología y estrategias didácticas**

Metodología Institucional:

a) Elaboración de ensayos, monografías e investigaciones (según el nivel) consultando fuentes bibliográficas, hemerográficas y en Internet.
b) Elaboración de reportes de lectura de artículos en lengua inglesa, actuales y relevantes.

Estrategias del Modelo UACJ Visión 2020 recomendadas para el curso:

a) búsqueda, organización y recuperación de información
b) evaluación
c) investigación
d) meta cognitivas
e) problematización
f) proceso de pensamiento lógico y crítico
g) procesamiento, apropiación-construcción
h) significación generalización
i) trabajo colaborativo

**IX. Criterios de evaluación y acreditación**

a) Institucionales de acreditación:

Acreditación mínima de 80% de clases programadas

Entrega oportuna de trabajos

Pago de derechos

Calificación ordinaria mínima de 7.0

Permite examen único: si

b) Evaluación del curso

Acreditación de los temas mediante los siguientes porcentajes:

| | |
|---|---|
| Tema 1 | 10% |
| Tema 2 | 10% |
| Tema 3 | 10% |
| Tema 4 | 10% |
| Tema 5 | 10% |
| Tareas | 10% |
| Prácticas | 20% |
| Proyecto | 20% |
| Total | 100 % |

## X. Bibliografía


- Stroustrup, B. (1997). The C++ Programming Language (Tercera edición y edición especial). Addison-Wesley. Estados Unidos de América.
- Stroustrup, B. (2008). Programming, principles and practice using C++. Addison-Wesley. Estado Unidos de América.
- Ruiz, D. (2004). C++ Programación Orientada a Objetos. Mega Punto ediciones. España.
- Smith, J. A. (1999). C++ Programación Orientada a Objetos. Paraninfo Cengage Learning. España.
- Smith J. A. (2001). Desarrollo de proyectos con programación orientada a objetos con C++. Paraninfo-México España.
- Ceballos Sierra, F. J. (2006). Programación orientada a objetos con C++, Cuarta Edición, Alfa Omega. Madrid.
- Jacobson, Rumbaugh & Booch, (1999). Unified modeling language. User Guide. Addison Wesley. Traducción al Espanol: Lenguaje de Modelado Unificado. Pearson Educación S. A. Madrid, España.


## X. Perfil deseable del docente

Doctorado o Maestría en áreas afines a Ciencias de la Computación y/o Tecnologías de Información; con experiencia en el manejo de lenguajes de programación y el paradigma de orientación a objetos. En su defecto, Ingeniero en Sistemas Computacionales o equivalente, con mínimo 2 años de experiencia en la industria, con experiencia en programación.

## XI. Institucionalización

Responsable del Departamento: Mtro. Jesús Armando Gándara

Coordinador/a del Programa: Ing. Cynthia Vanessa

Fecha de elaboración: 20 de enero, 2011

Elaboró: Dra. Leticia Ortega Máynez

Fecha de rediseño: 16 de mayo, 2011

Rediseño: Dra. Leticia Ortega Máynez