



Universidad de Ingeniería y Tecnología
Escuela Profesional de
Ciencia de la Computación
Silabo del curso – Periodo Académico 2017-II

1. Código del curso y nombre: CS112. Programación Orientada a Objetos I

2. Créditos: 4

3. Horas de Teoría y Laboratorio: 2 HT; 4 HP;

4. Docente(s)

Dr. Jose Antonio Fiestas Iquira

- Dr. Ciencias Naturales, UNIHEIDELBERG, Alemania, 2006.
- Mag. Física, UNIHEIDELBERG, Alemania, 2002.

Dr. Ernesto Cuadros-Vargas

- Dr. Ciencia de la Computación, ICMC-USP, Brasil, 2004.
- Mag. Ciencia de la Computación, ICMC-USP, Brasil, 1998.

Mg. Teófilo Chambilla Aquino

- Mag. Ciencias, UCHILE, Chile, 2015.

Mg. Jorge Luis Alvarado Revata

- Mag. Gestión de la Investigación, UNIA, España, 2016.

Prof. Carlo Christian Malpartida Anton

- Prof. Ingeniería Eléctrica, PUCP, Perú, 2012.

Atención previa coordinación con el profesor

5. Bibliografía

[PH13] Deitel. P.J and Deitel. H.M. *C++ How to Program (Early Objects Version)*. Deitel, How to Program. Prentice Hall, 2013. ISBN: 9780133378719. URL: <http://books.google.com.pe/books?id=XIZJNQEACAAJ>.

[Str13] Bjarne Stroustrup. *The C++ Programming Language*. 4th. Addison-Wesley, 2013. ISBN: 978-0-321-56384-2.

6. Información del curso

- (a) **Breve descripción del curso** Este es el segundo curso en la secuencia de los cursos introductorios a la informática. El curso servirá como puente entre el paradigma de la imperativo y el orientado al objeto, a demás introducirá a los participantes en los diversos temas del área de computación como: algoritmos, estructuras de datos, ingeniería del software, etc.
- (b) **Prerrequisitos:** CS111. Introducción a la Ciencia de la Computación. (1^{er} Sem)
- (c) **Tipo de Curso:** Obligatorio

7. Competencias

- Introducir al alumno a los fundamentos del paradigma de orientación a objetos, permitiendo asimilar los conceptos necesarios para desarrollar sistemas de información.

8. Contribución a los resultados (*Outcomes*)

- a) Aplicar conocimientos de computación y de matemáticas apropiadas para la disciplina. (**Evaluar**)
- c) Diseñar, implementar y evaluar un sistema, proceso, componente o programa computacional para alcanzar las necesidades deseadas. (**Evaluar**)

9. Competencias (IEEE)

- C1.** La comprensión intelectual y la capacidad de aplicar las bases matemáticas y la teoría de la informática (*Computer Science*).⇒ **Outcome a**
- C3.** Una comprensión intelectual de, y el aprecio por el papel central de los algoritmos y estructuras de datos.⇒ **Outcome c**
- CS1.** Modelar y diseñar sistemas de computadora de una manera que se demuestre comprensión del balance entre las opciones de diseño.⇒ **Outcome c**
- CS2.** Identificar y analizar los criterios y especificaciones apropiadas a los problemas específicos, y planificar estrategias para su solución.⇒ **Outcome c**

10. Lista de temas a estudiar en el curso

1. Visión General de los Lenguajes de Programación
2. Máquinas virtuales
3. Sistemas de tipos básicos
4. Conceptos Fundamentales de Programación
5. Programación orientada a objetos
6. Algoritmos y Diseño
7. Estrategias Algorítmicas
8. Análisis Básico
9. Algoritmos y Estructuras de Datos fundamentales

11. Metodología y Evaluación

Sesiones Teóricas:

El desarrollo de las sesiones teóricas está focalizado en el estudiante, a través de su participación activa, resolviendo problemas relacionados al curso con los aportes individuales y discutiendo casos reales de la industria. Los alumnos desarrollarán a lo largo del curso un proyecto de aplicación de las herramientas recibidas en una empresa.

Sesiones de Laboratorio:

Las sesiones prácticas se desarrollan en laboratorio. Las prácticas de laboratorio se realizan en equipos para fortalecer su comunicación. Al inicio de cada laboratorio se explica el desarrollo de la práctica y al término se destaca las principales conclusiones de la actividad en forma grupal.

Exposiciones individuales o grupales:

Se fomenta la participación individual y en equipo para exponer sus ideas, motivándolos con puntos adicionales en las diferentes etapas de la evaluación del curso.

Lecturas:

A lo largo del curso se proporcionan diferentes lecturas, las cuales son evaluadas. El promedio de las notas de las lecturas es considerado como la nota de una práctica calificada. El uso del campus virtual UTEC Online permite a cada estudiante acceder a la información del curso, e interactuar fuera de aula con el profesor y con los otros estudiantes.

Sistema de Evaluación:

$$NT = 6\%TE1 + 14\%TE2 + 16\%P1 + 24\%P2 + 4\%(SP1 + SP2 + SP3 + SP4 + SP5 + SP6 + SP7 + SP8)$$

Donde:

- TE_i = Evaluaciones Teóricas #i, $1 \leq i \leq 2$
- P_j = Evaluaciones Teóricas #i, $1 \leq j \leq 2$

- SP_k = Conjuntos de problemas $\#i$, $1 \leq k \leq 8$

12. Contenido

Unidad 1: Visión General de los Lenguajes de Programación (1)	
Competences esperadas: C1	
Objetivos de Aprendizaje	Tópicos
<ul style="list-style-type: none"> • Discutir el contexto histórico de los paradigmas de diversos lenguajes de programación [Familiarizarse] 	<ul style="list-style-type: none"> • Breve revisión de los paradigmas de programación. • Comparación entre programación funcional y programación imperativa. • Historia de los lenguajes de programación.
Lecturas : [Str13], [PH13]	

Unidad 2: Máquinas virtuales (1)	
Competences esperadas: C2	
Objetivos de Aprendizaje	Tópicos
<ul style="list-style-type: none"> • Explicar el concepto de memoria virtual y la forma cómo se realiza en hardware y software [Familiarizarse] • Diferenciar emulación y el aislamiento [Familiarizarse] • Evaluar virtualización de compensaciones [Evaluar] 	<ul style="list-style-type: none"> • El concepto de máquina virtual. • Tipos de virtualización (incluyendo Hardware / Software, OS, Servidor, Servicio, Red) . • Lenguajes intermedios.
Lecturas : [Str13], [PH13]	

Unidad 3: Sistemas de tipos básicos (2)	
Competences esperadas: C1,C2,CS1	
Objetivos de Aprendizaje	Tópicos
<ul style="list-style-type: none"> • Tanto para tipo primitivo y un tipo compuesto, describir de manera informal los valores que tiene dicho tipo [Familiarizarse] • Para un lenguaje con sistema de tipos estático, describir las operaciones que están prohibidas de forma estática, como pasar el tipo incorrecto de valor a una función o método [Familiarizarse] • Describir ejemplos de errores de programa detectadas por un sistema de tipos [Familiarizarse] • Para múltiples lenguajes de programación, identificar propiedades de un programa con verificación estática y propiedades de un programa con verificación dinámica [Usar] • Dar un ejemplo de un programa que no verifique tipos en un lenguaje particular y sin embargo no tenga error cuando es ejecutado [Familiarizarse] • Usar tipos y mensajes de error de tipos para escribir y depurar programas [Usar] • Explicar como las reglas de tipificación definen el conjunto de operaciones que legales para un tipo [Familiarizarse] • Escribir las reglas de tipo que rigen el uso de un particular tipo compuesto [Usar] • Explicar por qué indecidibilidad requiere sistemas de tipo para conservadoramente aproximar el comportamiento de un programa [Familiarizarse] • Definir y usar piezas de programas (tales como, funciones, clases, métodos) que usan tipos genéricos, incluyendo para colecciones [Usar] • Discutir las diferencias entre, genéricos (<i>generics</i>), subtipo y sobrecarga [Familiarizarse] • Explicar múltiples beneficios y limitaciones de tipificación estática en escritura, mantenimiento y depuración de un software [Familiarizarse] 	<ul style="list-style-type: none"> • Tipos como conjunto de valores junto con un conjunto de operaciones. <ul style="list-style-type: none"> – Tipos primitivos (p.e. numeros, booleanos) – Composición de tipos contruidos de otros tipos (p.e., registros, uniones, arreglos, listas, funciones, referencias) • Declaración de modelos (enlace, visibilidad, alcance y tiempo de vida). • Vista general del chequeo de tipos.
Lecturas : [Str13], [PH13]	

Unidad 4: Conceptos Fundamentales de Programación (6)	
Competences esperadas: C1,C2,CS2	
Objetivos de Aprendizaje	Tópicos
<ul style="list-style-type: none"> • Analiza y explica el comportamiento de programas simples que involucran estructuras fundamentales de programación variables, expresiones, asignaciones, E/S, estructuras de control, funciones, paso de parámetros, y recursividad [Evaluar] • Identifica y describe el uso de tipos de datos primitivos [Familiarizarse] • Escribe programas que usan tipos de datos primitivos [Usar] • Modifica y expande programas cortos que usen estructuras de control condicionales e iterativas así como funciones [Usar] • Diseña, implementa, prueba, y depura un programa que usa cada una de las siguientes estructuras de datos fundamentales: cálculos básicos, E/S simple, condicional estándar y estructuras iterativas, definición de funciones, y paso de parámetros [Usar] • Escribe un programa que usa E/S de archivos para brindar persistencia a través de ejecuciones múltiples [Usar] • Escoje estructuras de condición y repetición adecuadas para una tarea de programación dada [Evaluar] • Describe el concepto de recursividad y da ejemplos de su uso [Familiarizarse] • Identifica el caso base y el caso general de un problema basado en recursividad [Evaluar] 	<ul style="list-style-type: none"> • Sintaxis y semántica básica de un lenguaje de alto nivel. • Variables y tipos de datos primitivos (ej., numeros, caracteres, booleanos) • Expresiones y asignaciones. • Operaciones básicas I/O incluyendo archivos I/O. • Estructuras de control condicional e iterativas. • Paso de funciones y parámetros.
Lecturas : [Str13], [PH13]	

Unidad 5: Programación orientada a objetos (10)	
Competences esperadas: C2,C24,CS1,CS2	
Objetivos de Aprendizaje	Tópicos
<ul style="list-style-type: none"> • Diseñar e implementar una clase [Usar] • Usar subclase para diseñar una jerarquía simple de clases que permita al código ser reusable por diferentes subclases [Usar] • Razonar correctamente sobre el flujo de control en un programa mediante el envío dinámico [Usar] • Comparar y contrastar (1) el enfoque procedurar/funcional- definiendo una función por cada operación con el cuerdo de la función proporcionando un caso por cada variación de dato - y (2) el enfoque orientado a objetos - definiendo una clase por cada variación de dato con la definición de la clase proporcionando un método por cada operación. Entender ambos enfoques como una definición de variaciones y operaciones de una matriz [Evaluar] • Explicar la relación entre la herencia orientada a objetos (codigo compartido y <i>overriding</i>) y subtipificación (la idea de un subtipo es ser utilizable en un contexto en el que espera al supertipo) [Familiarizarse] • Usar mecanismos de encapsulación orientada a objetos, tal como interfaces y miembros privados [Usar] • Definir y usar iteradores y otras operaciones sobre agregaciones, incluyendo operaciones que tienen funciones como argumentos, en múltiples lenguajes de programación, seleccionar la forma mas natural por cada lenguaje [Usar] 	<ul style="list-style-type: none"> • Diseño orientado a objetos: <ul style="list-style-type: none"> – Descomposicion en objetos que almacenan estados y poseen comportamiento – Diseño basado en jerarquia de clases para modelamiento • Lenguajes orientados a objetos para la encapsulación: <ul style="list-style-type: none"> – privacidad y la visibilidad de miembros de la clase – Interfaces revelan único método de firmas – clases base abstractas • Definición de las categorías, campos, métodos y constructores. • Las subclases, herencia y método de alteración temporal. • Subtipificación: <ul style="list-style-type: none"> – Polimorfismo artículo Subtipo; upcasts implícitos en lenguajes con tipos. – Noción de reemplazo de comportamiento: los subtipos de actuar como supertipos. – Relación entre subtipos y la herencia. • Uso de coleccion de clases, iteradores, y otros componentes de la libreria estandar. • Asignación dinámica: definición de método de llamada.
Lecturas : [Str13], [PH13]	

Unidad 6: Algoritmos y Diseño (3)	
Competences esperadas: C1,C2,C23	
Objetivos de Aprendizaje	Tópicos
<ul style="list-style-type: none"> • Discute la importancia de los algoritmos en el proceso de solución de un problema [Familiarizarse] • Discute como un problema puede ser resuelto por múltiples algoritmos, cada uno con propiedades diferentes [Familiarizarse] • Crea algoritmos para resolver problemas simples [Usar] • Usa un lenguaje de programación para implementar, probar, y depurar algoritmos para resolver problemas simples [Usar] • Implementa, prueba, y depura funciones recursivas simples y sus procedimientos [Usar] • Determina si una solución iterativa o recursiva es la más apropiada para un problema [Evaluar] • Implementa un algoritmo de divide y vencerás para resolver un problema [Usar] • Aplica técnicas de descomposición para dividir un programa en partes más pequeñas [Usar] • Identifica los componentes de datos y el comportamiento de múltiples tipos de datos abstractos [Usar] • Implementa un tipo de dato abstracto coherente, con la menor pérdida de acoplamiento entre componentes y comportamientos [Usar] • Identifica las fortalezas y las debilidades relativas entre múltiples diseños e implementaciones de un problema [Evaluar] 	<ul style="list-style-type: none"> • Estrategias de solución de problemas <ul style="list-style-type: none"> – Funciones matemáticas iterativas y recursivas – Recorrido iterativo y recursivo en estructura de datos – Estrategias Divide y Conquistar • Rol de los algoritmos en el proceso de solución de problemas • Estrategias de solución de problemas <ul style="list-style-type: none"> – Funciones matemáticas iterativas y recursivas – Recorrido iterativo y recursivo en estructura de datos – Estrategias Divide y Conquistar • Conceptos y principios fundamentales de diseño <ul style="list-style-type: none"> – Abstracción – Descomposición de Program – Encapsulamiento y camuflaje de información – Separación de comportamiento y aplicación
Lecturas : [Str13], [PH13]	

Unidad 7: Estrategias Algorítmicas (3)	
Competences esperadas: C1,C2,C24,CS1	
Objetivos de Aprendizaje	Tópicos
<ul style="list-style-type: none"> • Para cada una de las estrategias (fuerza bruta, algoritmo goloso, divide y vencerás, recursividad en reversa y programación dinámica), identifica un ejemplo práctico en el cual se pueda aplicar [Familiarizarse] • Utiliza un enfoque voraz para resolver un problema específico y determina si la regla escogida lo guía a una solución óptima [Evaluar] • Usa un algoritmo de divide-y-vencerás para resolver un determinado problema [Usar] • Usa recursividad en reversa a fin de resolver un problema como en el caso de recorrer un laberinto [Usar] • Usa programación dinámica para resolver un problema determinado [Usar] • Determina el enfoque algorítmico adecuado para un problema [Evaluar] • Describe varios métodos basados en heurísticas para resolver problemas [Familiarizarse] 	<ul style="list-style-type: none"> • Algoritmos de fuerza bruta. • Algoritmos voraces. • Divide y vencerás. • Backtracking recursivo. • Programación Dinámica.
Lecturas : [Str13], [PH13]	

Unidad 8: Análisis Básico (2)	
Competences esperadas: C1,C2,C24,CS1	
Objetivos de Aprendizaje	Tópicos
<ul style="list-style-type: none"> • Explique a que se refiere con “mejor”, “esperado” y “peor” caso de comportamiento de un algoritmo [Familiarizarse] 	<ul style="list-style-type: none"> • Diferencias entre el mejor, el esperado y el peor caso de un algoritmo.
Lecturas : [Str13], [PH13]	

Unidad 9: Algoritmos y Estructuras de Datos fundamentales (6)	
Competences esperadas: C1,C2,C24,CS1	
Objetivos de Aprendizaje	Tópicos
<ul style="list-style-type: none"> • Implementar algoritmos numéricos básicos [Usar] • Implementar algoritmos de búsqueda simple y explicar las diferencias en sus tiempos de complejidad [Evaluar] • Ser capaz de implementar algoritmos de ordenamiento comunes cuadráticos y $O(N \log N)$ [Usar] • Describir la implementación de tablas hash, incluyendo resolución y el evitamiento de colisiones [Familiarizarse] • Discutir el tiempo de ejecución y eficiencia de memoria de los principales algoritmos de ordenamiento, búsqueda y hashing [Familiarizarse] • Discutir factores otros que no sean eficiencia computacional que influyan en la elección de algoritmos, tales como tiempo de programación, mantenibilidad, y el uso de patrones específicos de la aplicación en los datos de entrada [Familiarizarse] • Explicar como el balanceamiento del arbol afecta la eficiencia de varias operaciones de un arbol de búsqueda binaria [Familiarizarse] • Resolver problemas usando algoritmos básicos de grafos, incluyendo búsqueda por profundidad y búsqueda por amplitud [Usar] • Demostrar habilidad para evaluar algoritmos, para seleccionar de un rango de posibles opciones, para proveer una justificación por esa selección,y para implementar el algoritmo en un contexto en específico [Evaluar] • Describir la propiedad del heap y el uso de heaps como una implementación de colas de prioridad [Familiarizarse] • Resolver problemas usando algoritmos de grafos, incluyendo camino más corto de una sola fuente y camino más corto de todos los pares, y como mínimo un algoritmo de arbol de expansion minima [Usar] • Trazar y/o implementar un algoritmo de comparación de string [Usar] 	<ul style="list-style-type: none"> • Algoritmos numéricos simples, tales como el cálculo de la media de una lista de números, encontrar el mínimo y máximo. • Algoritmos de búsqueda secuencial y binaria. • Algoritmos de ordenamiento de peor caso cuadrático (selección, inserción) • Algoritmos de ordenamiento con peor caso o caso promedio en $O(N \lg N)$ (Quicksort, Heapsort, Mergesort) • Tablas Hash, incluyendo estrategias para evitar y resolver colisiones. • Árboles de búsqueda binaria: <ul style="list-style-type: none"> – Operaciones comunes en árboles de búsqueda binaria como seleccionar el mínimo, máximo, insertar, eliminar, recorrido en árboles. • Grafos y algoritmos en grafos: <ul style="list-style-type: none"> – Representación de grafos (ej., lista de adyacencia, matriz de adyacencia) – Recorrido en profundidad y amplitud
Lecturas : [Str13], [PH13]	