

Guía docente

230086 - POO - Programación Orientada a Objetos

Última modificación: 11/06/2026

Unidad responsable: Escuela Técnica Superior de Ingeniería de Telecomunicación de Barcelona
Unidad que imparte: 701 - DAC - Departamento de Arquitectura de Computadores.

Titulación: GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN (Plan 2015). (Asignatura obligatoria).

Curso: 2026 **Créditos ECTS:** 6.0 **Idiomas:** Catalán, Castellano

PROFESORADO

Profesorado responsable: JUAN CARLOS CRUELLAS IBARZ

Otros: Primer quadrimestre:
JUAN CARLOS CRUELLAS IBARZ - 11
MARIA LUISA GIL GOMEZ - 11, 13
JORDI PERELLÓ MUNTAN - 11, 12, 13

COMPETENCIAS DE LA TITULACIÓN A LAS QUE CONTRIBUYE LA ASIGNATURA

Transversales:

06 URI N1. USO SOLVENTE DE LOS RECURSOS DE INFORMACIÓN - Nivel 1: Identificar las propias necesidades de información y utilizar las colecciones, los espacios y los servicios disponibles para diseñar y ejecutar búsquedas simples adecuadas al ámbito temático.

METODOLOGÍAS DOCENTES

- Clases expositivas
- Clases de laboratorio

La asistencia a las clases de laboratorio es obligatoria. Por cada ausencia injustificada a una clase de laboratorio se descontarán 0,5 puntos de la nota de laboratorio.

OBJETIVOS DE APRENDIZAJE DE LA ASIGNATURA

Resultado del aprendizaje:

El estudiante entiende en qué consiste el paradigma de la programación orientada a objetos.

El estudiante entiende los conceptos fundamentales: clase, objeto, encapsulación, agregación, herencia i polimorfismo y puede demostrarlo generando código que defina clases de Java, las instancie, incluya contenedores como atributos de clases e implemente correctamente subclases y clases que implementen un Interface de Java..

El estudiante consolida y amplía los algoritmos de ordenación y búsqueda mostrados en 1A en lenguaje Java, y es capaz de demostrarlo implementándolos en Java.

El estudiante es capaz de generar un programa Java a partir de un diseño definido mediante diagramas de clases UML y la especificación de los métodos de todas sus clases.

HORAS TOTALES DE DEDICACIÓN DEL ESTUDIANTE

Tipo	Horas	Porcentaje
Horas grupo pequeño	26,0	17.33
Horas aprendizaje autónomo	85,0	56.67
Horas grupo grande	39,0	26.00

Dedicación total: 150 h

CONTENIDOS

Tema 1. El paradigma de la orientación a objetos: clases y objetos

Descripción:

Presentación del paradigma de la orientación a objetos. Conceptos de objeto y clase. Atributos y métodos.

Objetivos específicos:

El estudiante debe comprender el concepto de objeto como entidad que aglutina datos (atributos) y funciones (métodos).
El estudiante debe comprender el concepto de clase como plantilla que define los atributos y métodos de todos los objetos instancias de esa clase.

Dedicación: 2h

Aprendizaje autónomo: 2h

Tema 2. Clases y objetos en Java

Descripción:

Sintaxis de definición de clases en Java.

Definición de atributos. Modificadores de acceso: public, private.

Definición de métodos. Métodos especiales: constructores, métodos de acceso a atributos (set y get).

Creación de objetos: operador new. Gestión dinámica de memoria

Atributos y métodos estáticos. Su razón de ser y su uso.

Modificadores constant y final.

Objetivos específicos:

El estudiante debe aprender la sintaxis de definición de clases en Java, incluyendo la sintaxis de definición de los diferentes tipos de atributos y la sintaxis de definición de los métodos de las clases.

El estudiante debe aprender los diferentes tipos de modificadores de acceso, sus diferencias y usos.

El estudiante debe ser capaz de discriminar entre atributos/métodos de clase (static) y atributos/métodos de objeto (no static), y debe ser capaz de utilizar unos y otros convenientemente.

El estudiante debe demostrar lo anterior siendo capaz de definir correctamente clases de Java con los diferentes tipos de atributos y métodos, afectados por los correspondientes modificadores.

Dedicación: 21h

Grupo grande/Teoría: 6h

Aprendizaje autónomo: 15h

Tema 3. Tipos primitivos en Java y clases relevantes

Descripción:

Tipos primitivos en Java: tipos de enteros, tipos de reales, tipo caracter (diferencia con respecto a C), booleano, byte.

Clases correspondientes a los tipos primitivos: Integer, Double, Float, Character, Boolean, Byte.

Clase String. Concepto y métodos más relevantes.

De valores numéricos a su representación textual como Strings y viceversa.

Objetivos específicos:

El estudiante debe conocer todos los tipos primitivos definidos en Java, así como las principales diferencias con los tipos predefinidos en C.

El estudiante debe conocer las clases Java que encapsulan los tipos primitivos.

El estudiante debe saber realizar las conversiones valores numéricos -> representaciones textuales de dichos valores y viceversa, haciendo uso de los métodos estáticos de String y las clases que encapsulan a los tipos numéricos primitivos.

El estudiante debe ser capaz de demostrar lo anterior declarando correctamente variables de los tipos primitivos, y programando correctamente las conversiones valores numéricos <-> representaciones textuales.

Dedicación: 7h

Grupo grande/Teoría: 2h

Aprendizaje autónomo: 5h

Tema 4. Contenedores

Descripción:

Concepto de objeto contenedor.

Tipos de contenedores: secuencias, diccionarios y conjuntos.

Implementación de secuencias en Java: ArrayList y LinkedList. Métodos más relevantes.

Iteradores. Concepto y su papel en el recorrido de secuencias.

Implementación de diccionarios en Java: HashMap. Métodos más relevantes.

Obtención de vistas parciales de un diccionario: métodos keySet(), values()

Implementación de conjuntos en Java: HashSet. Métodos más relevantes.

Identificación del tipo de contenedor que es preciso utilizar en cada caso.

Objetivos específicos:

El estudiante debe saber declarar variables de cualquier clase contenedor en Java.

El estudiante debe saber crear un contenedor vacío.

El estudiante debe saber utilizar correctamente los métodos que los contenedores ponen a su disposición para usarlos de la mejor manera posible.

El estudiante debe, a la vista de los requisitos de un problema, saber seleccionar el tipo de contenedor adecuado a dichos requisitos (mapa, lista o conjunto).

El estudiante debe saber realizar recorridos totales o parciales sobre los objetos depositados en un contenedor.

El estudiante debe saber definir atributos de clase como contenedores y debe saber gestionar correctamente dichos atributos.

El estudiante debe demostrar lo anterior siendo capaz generar código Java que defina objetos contenedores, los cree, y vaya gestionando su llenado, recorrido y eventual vaciado, en función de una especificación completa de un programa.

Dedicación: 32h

Grupo grande/Teoría: 8h

Aprendizaje autónomo: 24h

Tema 5. Introducción al UML. Diagrama de clases

Descripción:

El UML como un lenguaje de representación gráfica de un programa.

El diagrama de clases.

Relaciones entre clases: asociaciones y dependencias.

Definición completa de una asociación: nombre, direccionalidad, cardinalidad.

Implementación de las asociaciones en el código Java.

Objetivos específicos:

El estudiante debe comprender la semántica de los tres tipos de relaciones: herencia, asociación y dependencia.

El estudiante debe saber identificar los diferentes tipos de relaciones a partir de sus respectivas representaciones gráficas.

El estudiante debe ser capaz de identificar la direccionalidad y la cardinalidad de las asociaciones.

El estudiante debe conocer la sintaxis UML para representar atributos y cabeceras de métodos de una clase.

El estudiante debe conocer la sintaxis UML para la representación de clases abstractas e interfaces.

El estudiante debe conocer la sintaxis UML para denotar que una clase implementa un cierto Interface Java.

El estudiante debe ser capaz de implementar las asociaciones direccionales con atributos. Debe ser capaz también de implementar asociaciones de uno a muchos con contenedores.

El estudiante debe demostrar todo lo anterior generando los atributos de un grupo de clases a partir de un diagrama de clases UML, en sintaxis Java.

Dedicación: 11h

Grupo grande/Teoría: 3h

Aprendizaje autónomo: 8h

Tema 6. Herencia y polimorfismo

Descripción:

Concepto de herencia. Superclases y clases derivadas (subclases). Herencia de atributos y métodos. Constructores en las subclases.

La herencia como relación entre clases. Representación de la herencia en diagramas de clase. Los objetos instancia de una subclase son también objetos instancia de la superclase.

La clase Object como clase raíz de la jerarquía de clases en Java. El método toString() de Object.

Concepto de polimorfismo. Métodos polimórficos.

El Interface en Java. El interface como simulador de herencia múltiple. El interface como mecanismo que permite a un mismo objeto presentar varios tipos de comportamiento.

Ordenación y búsqueda en vectores polimórficos.

Objetivos específicos:

El estudiante debe comprender que la herencia es una relación "es un tipo específico de" y todas sus implicaciones.

El estudiante debe comprender que un objeto instancia de una clase inmersa en una jerarquía de clases es a la vez un objeto instancia de todas las clases ancestro que se hallan en esta jerarquía.

El estudiante debe comprender qué implica que un atributo o una variable local se declaren como instancia de una superclase.

El estudiante debe comprender qué implica que un método retorne instancias de una superclase.

El estudiante debe comprender qué implica que una clase implemente uno o varios interfaces Java.

El estudiante debe comprender qué es el polimorfismo y sus implicaciones.

El estudiante debe comprender las ventajas que supone la programación contra lo más genérico.

El estudiante debe probar lo anterior generando código Java que defina e implemente correctamente jerarquías de clases y soluciones polimórficas cuando sean requeridas.

Dedicación: 22h

Grupo grande/Teoría: 9h

Aprendizaje autónomo: 13h

Tema 7. Excepciones

Descripción:

El estudiante debe comprender el concepto de excepción como un objeto especial "lanzado" por un método cuando algo inusual ocurre.

El estudiante debe comprender las diferencias entre excepciones explícitas (checked) y excepciones implícitas (non-checked).

El estudiante debe conocer la sintaxis Java de definición de clases excepción.

El estudiante debe conocer la sintaxis Java que declara que un método puede lanzar excepciones explícitas (checked).

El estudiante debe conocer la sintaxis Java de captura de excepciones.

El estudiante debe probar lo anterior siendo capaz de generar métodos que, bajo determinadas circunstancias, creen y lancen excepciones, siendo capaz de decidir dónde capturar dichas excepciones lanzadas, y siendo capaz de generar el código pertinente para el tratamiento de las excepciones capturadas.

Objetivos específicos:

El concepto de excepción. Generación, lanzamiento y recogida de excepciones en Java.

Tratamiento de excepciones.

Dedicación: 11h 30m

Grupo grande/Teoría: 4h 30m

Aprendizaje autónomo: 7h

Tema 8. Entrada/Salida en Java

Descripción:

Concepto de flujo de datos ("stream").

Jerarquía de clases Java per Entrada/Salida para flujos de caracteres

Clases para Entrada/Salida de caracteres desde/hacia archivos.

Objetivos específicos:

El estudiante debe entender la jerarquía de clases cuya raíz es InputStream.

El estudiante debe entender la clase FileInputStream como clase especializada en leer bytes de un archivo.

El estudiante debe entender cómo las clases Scanner y FileInputStream colaboran para leer un archivo de texto estructurado en líneas.

El El estudiante debe entender la jerarquía de clases cuya raíz es OutputStream.

El estudiante debe entender la clase FileOutputStream como clase especializada en leer bytes de un archivo.

El estudiante debe entender la clase PrintStream como clase especializada en tomar bytes y escribir caracteres en un objeto de una clase derivada de OutputStream.

El estudiante debe demostrar lo anterior siendo capaz de generar código que permita leer archivos de texto estructurados en líneas, en los que cada línea, además, se dota de una estructura interna determinada, y procesar los contenidos de cada línea según lo especificado en un diseño.

El estudiante debe demostrar lo anterior siendo capaz de generar código que permita depositar en un archivo de texto estructurado en líneas dotadas de una cierta estructura interna, la información de uno o varios objetos, tal y como se especifique en un diseño.

Dedicación: 4h

Grupo grande/Teoría: 1h 30m

Aprendizaje autónomo: 2h 30m



ACTIVIDADES

PRÁCTICA DE LABORATORIO 1: ENTORNO, DEPURACIÓN Y ALGORÍTMOS BÁSICOS

Descripción:

Introducción al entorno NetBeans. Creación de proyectos. Edición de código. Depuración. Implementación de algoritmos básicos.

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h

PRUEBAS DE RESPUESTA CORTA (CONTROL)

Descripción:

Prueba de consolidación de conocimientos

Dedicación: 2h

Grupo grande/Teoría: 2h

PRÁCTICA DE LABORATORIO 2: ARRAYS Y ALGORÍTMOS DE ORDENACIÓN.

Descripción:

Uso de vectores en Java.

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h

PRÁCTICA DE LABORATORIO 3: ALGORÍTMOS DE BÚSQUEDA

Descripción:

Uso de objetos en Java: String y StringBuffer. Sesión con métodos relevantes de String: split(), indexOf(), etc

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h

PRÁCTICA DE LABORATORIO 4: IMPLEMENTACIÓN DE CLASES Y USO DE OBJETOS EN JAVA.

Descripción:

Definición e instanciación de una clase.

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h

PRÁCTICA DE LABORATORIO 5: ALGORÍTMOS PARA USO DE LISTAS, DICCIONARIOS Y CONJUNTOS EN JAVA.

Descripción:

Usando los contenedores en Java.

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h



PRÁCTICA DE LABORATORIO 6: INICIO DE CONSTRUCCIÓN DE PEQUEÑA APLICACIÓN; EL LABERINTO (1)

Descripción:

Primer paso para construir una pequeña aplicación con contenedores, herencia, polimorfismo, excepciones y entrada/salida (1)

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h

PRÁCTICA DE LABORATORIO 7: HERENCIA Y POLIMORFISMO, EL LABERINTO (2)

Descripción:

Segunda versión de una pequeña aplicación incorporando herencia y polimorfismo (2)

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h

PRÁCTICA DE LABORATORIO 8: EXCEPCIONES EN EL LABERINTO (3)

Descripción:

Implementación de una pequeña aplicación incorporando excepciones (3)

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h

PRUEBAS DE RESPUESTA LARGA (EXAMEN FINAL)

Dedicación: 3h

Grupo pequeño/Laboratorio: 3h

PRÁCTICA DE LABORATORIO: ENTRADA/SALIDA EN JAVA

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h

EXAMEN DE LABORATORIO

Dedicación: 2h

Grupo pequeño/Laboratorio: 2h

SISTEMA DE CALIFICACIÓN

Cualquier acto de fraude académico, plagio o uso o mera tenencia al alcance de medios no autorizados en cualquier actividad de evaluación comportará la calificación de cero (0) en la prueba o entrega afectada. Además, de acuerdo con la normativa de la Universidad, la posible derivación de los hechos para la apertura de un expediente disciplinario implicará que la asignatura quede en el estado provisional de "pendiente de evaluación" hasta la resolución del expediente. La gestión de estas incidencias se lleva a cabo de acuerdo con el [Marco de actuación para la integridad académica en la evaluación de la UPC](#).

- Examen(es) de laboratorio 25%
- Prueba corta 20%
- Examen final 55%

La asistencia a las clases de laboratorio es obligatoria. Por cada ausencia injustificada a una clase de laboratorio se descontarán 0,5 puntos de la nota de laboratorio.

NORMAS PARA LA REALIZACIÓN DE LAS PRUEBAS.

El(los) examen(es) de laboratorio y la prueba corta no se revalúan.

La asistencia a las clases de laboratorio es obligatoria. Por cada ausencia injustificada a una clase de laboratorio se descontarán 0,5 puntos de la nota de laboratorio.

BIBLIOGRAFÍA

Básica:

- Eckel, Bruce. Piensa en Java. 4a ed. Madrid: Prentice Hall, 2007. ISBN 9788489660342.
- García de Jalón, J. [i 7 més]. Aprende Java como si estuviera en primero [en línea]. San Sebastián: Escuela Superior de Ingenieros Industriales. Universidad de Navarra, 2000 [Consulta: 07/07/2020]. Disponible a: <http://ocw.uc3m.es/cursos-archivados/programacion-java/manuales/java2-U-Navarra.pdf/view>.

Complementaria:

- JAVA platform: API specification [en línea]. Standard Edition 8. Redwood Shores, CA: Oracle, 2020 [Consulta: 07/07/2020]. Disponible a: <https://docs.oracle.com/javase/8/docs/api/>.
- The JAVA tutorials [en línea]. Redwood Shores, CA: Oracle, 2019 [Consulta: 07/07/2020]. Disponible a: <https://docs.oracle.com/javase/tutorial/>.

RECURSOS

Material informático:

- Java Development Kit. Java Development Kit
- <https://netbeans.org/>. NetBeans