

CONTENIDO

| | |
|---------------|------|
| Prólogo | xvii |
|---------------|------|

Parte I

EL MUNDO DE LA ORIENTACION A OBJETOS: CONCEPTOS, RELACIONES, MODELADO Y LENGUAJES DE PROGRAMACION

| | |
|---|----------|
| Capítulo 1. El desarrollo de software | 3 |
| 1.1. La complejidad inherente al software | 4 |
| 1.1.1. La complejidad del dominio del problema | 4 |
| 1.1.2. La dificultad de gestionar el proceso de desarrollo | 4 |
| 1.1.3. La flexibilidad a través del software | 5 |
| 1.2. La crisis del software | 5 |
| 1.3. Factores en la calidad del software | 7 |
| 1.3.1. Razones fundamentales que están influyendo en la importancia de la POO | 9 |
| 1.4. Programación y abstracción | 9 |
| 1.5. El papel (el rol) de la abstracción | 10 |
| 1.5.1. La abstracción como proceso natural mental | 10 |
| 1.5.2. Historia de la abstracción del software | 11 |
| 1.5.3. Procedimientos | 12 |
| 1.5.4. Módulos | 13 |
| 1.5.5. Tipos abstractos de datos | 13 |
| 1.5.6. Objetos | 14 |
| 1.6. Un nuevo paradigma de programación | 15 |
| 1.7. Orientación a objetos | 16 |
| 1.7.1. Abstracción | 17 |
| 1.7.2. Encapsulación | 18 |
| 1.7.3. Modularidad | 18 |
| 1.7.4. Jerarquía | 18 |
| 1.7.5. Polimorfismo | 19 |
| 1.7.6. Otras propiedades | 20 |
| 1.8. Reutilización de software | 21 |
| 1.9. Lenguajes de programación orientados a objetos | 22 |
| 1.9.1. Clasificación de los lenguajes orientados a objetos | 23 |

| | |
|--|-----------|
| 1.10. Desarrollo tradicional frente a orientado a objetos | 25 |
| 1.11. Beneficios de las tecnologías de objetos (TO) | 27 |
| Resumen | 29 |
| | |
| Capítulo 2. Modularidad: tipos abstractos de datos | 30 |
| 2.1. Modularidad | 31 |
| 2.1.1. La estructura de un módulo | 31 |
| 2.1.2. Reglas de modularización | 32 |
| 2.2. Diseño de módulos | 35 |
| 2.2.1. Acoplamiento de módulos | 35 |
| 2.2.2. Cohesión de módulos | 35 |
| 2.3. Tipos de datos | 36 |
| 2.4. Abstracción en lenguajes de programación | 38 |
| 2.4.1. Abstracciones de control | 38 |
| 2.4.2. Abstracción de datos | 39 |
| 2.5. Tipos abstractos de datos | 40 |
| 2.5.1. Ventajas de los tipos abstractos de datos | 42 |
| 2.5.2. Implementación de los TAD | 42 |
| 2.6. Tipos abstractos de datos en Turbo Pascal | 43 |
| 2.6.1. Aplicación del tipo abstracto de dato <i>Pila</i> | 45 |
| 2.7. Tipos abstractos de datos en Modula-2 | 46 |
| 2.7.1. Módulos | 46 |
| 2.7.2. Módulos locales | 47 |
| 2.7.3. Tipos opacos | 47 |
| 2.7.4. Tipos transparentes | 48 |
| 2.7.5. Una versión del tipo abstracto de dato <i>Pila</i> con datos opacos | 49 |
| 2.7.6. Otra aplicación del TAD <i>Pila</i> | 51 |
| 2.8. Tipos abstractos de datos en Ada | 53 |
| 2.8.1. Tipos privados | 55 |
| 2.8.2. Tipos privados limitados | 56 |
| 2.9. Tipos abstractos de datos en C | 57 |
| 2.9.1. Un ejemplo de un tipo abstracto de datos en C | 58 |
| 2.10. Tipos abstractos de datos en C++ | 60 |
| 2.10.1. Definición de una clase <i>Pila</i> en C++ | 61 |
| Resumen | 64 |
| Ejercicios | 65 |
| | |
| Capítulo 3. Conceptos fundamentales de programación orientada a objetos | 66 |
| 3.1. Programación estructurada | 67 |
| 3.1.1. Desventajas de la programación estructurada | 69 |
| 3.2. ¿Qué es la programación orientada a objetos? | 69 |
| 3.2.1. El objeto | 70 |
| 3.2.2. Ejemplos de objetos | 71 |
| 3.2.3. Métodos y mensajes | 73 |
| 3.3. Clases | 75 |
| 3.3.1. Implementación de clases en lenguajes | 75 |
| 3.3.2. Sintaxis | 76 |
| 3.4. Un mundo de objetos | 77 |

| | | |
|---------|--|-----|
| 3.4.1. | Definición de objetos | 78 |
| 3.4.2. | Identificación de objetos | 78 |
| 3.4.3. | Duración de los objetos | 80 |
| 3.4.4. | Objetos frente a clases. Representación gráfica (Notación de Ege). | 80 |
| 3.4.5. | Datos internos | 83 |
| 3.4.6. | Ocultación de datos | 84 |
| 3.5. | Herencia | 84 |
| 3.5.1. | Sintaxis | 86 |
| 3.5.2. | Tipos de herencia | 87 |
| 3.6. | Comunicaciones entre objetos: los mensajes | 89 |
| 3.6.1. | Activación de objetos | 90 |
| 3.6.2. | Mensajes | 90 |
| 3.6.3. | Paso de mensajes | 92 |
| 3.7. | Estructura interna de un objeto | 92 |
| 3.7.1. | Atributos | 93 |
| 3.7.2. | Métodos | 93 |
| 3.8. | Clases | 94 |
| 3.8.1. | Una comparación con tablas de datos | 95 |
| 3.9. | Herencia y tipos | 96 |
| 3.9.1. | Herencia simple (<i>herencia jerárquica</i>) | 98 |
| 3.9.2. | Herencia múltiple (<i>herencia en malla</i>) | 98 |
| 3.9.3. | Clases abstractas | 100 |
| 3.10. | Anulación/Sustitución | 101 |
| 3.11. | Sobrecarga | 102 |
| 3.12. | Ligadura dinámica | 104 |
| 3.12.1. | Funciones o métodos virtuales | 104 |
| 3.12.2. | Polimorfismo | 105 |
| 3.13. | Objetos compuestos | 105 |
| 3.13.1. | Un ejemplo de objetos compuestos | 107 |
| 3.13.2. | Niveles de profundidad | 107 |
| 3.14. | Reutilización con orientación a objetos | 109 |
| 3.14.1. | Objetos y reutilización | 110 |
| 3.15. | Polimorfismo | 110 |
| Resumen | | 111 |

Capítulo 4. Lenguajes de programación orientados a objetos 113

| | | |
|--------|---|-----|
| 4.1. | Evolución de los LPOOS | 114 |
| 4.1.1. | Estado actual de los lenguajes orientados a objetos en la década de los noventa | 117 |
| 4.2. | Clasificación de lenguajes orientados a objetos | 118 |
| 4.2.1. | Taxonomía de lenguajes orientados a objetos | 119 |
| 4.2.2. | Características de los lenguajes orientados a objetos | 120 |
| 4.2.3. | Puros frente a híbridos | 121 |
| 4.2.4. | Tipificación estática frente a dinámica | 122 |
| 4.2.5. | Ligadura estática frente a dinámica | 124 |
| 4.2.6. | Revisión de lenguajes orientados a objetos | 125 |
| 4.3. | Ada | 126 |
| 4.3.1. | Abstracción de datos en Ada | 126 |
| 4.3.2. | Genericidad en Ada | 127 |
| 4.3.3. | Sòpporte de herencia en Ada-83 | 128 |
| 4.3.4. | Sòpporte Ada para orientación a objetos | 128 |

| | | |
|--------|--|-----|
| 4.4. | Eiffel | 129 |
| 4.4.1. | La biblioteca de clases Eiffel | 130 |
| 4.4.2. | El entorno de programación Eiffel | 130 |
| 4.4.3. | El lenguaje Eiffel | 131 |
| 4.5. | Smalltalk | 132 |
| 4.5.1. | El lenguaje Smalltalk | 133 |
| 4.5.2. | La jerarquía de clases Smalltalk | 134 |
| 4.6. | Otros lenguajes de programación orientados a objetos | 134 |
| | Resumen | 135 |
| | Ejercicios | 136 |

Capítulo 5. Modelado de objetos: relaciones 137

| | | |
|----------|--|-----|
| 5.1. | Relaciones entre clases | 138 |
| 5.2. | Relación de generalización/especialización (<i>is-a/es-un</i>) | 138 |
| 5.2.1. | Jerarquías de generalización/especialización | 141 |
| 5.3. | Relación de agregación (<i>has-a/tiene-un</i>)..... | 143 |
| 5.3.1. | Agregación frente a generalización | 145 |
| 5.4. | Relación de asociación | 146 |
| 5.4.1. | Otros ejemplos de cardinalidad | 149 |
| 5.5. | Herencia: jerarquía de clases | 150 |
| 5.5.1. | Herencia simple | 151 |
| 5.5.2. | Herencia múltiple | 152 |
| 5.5.2.1. | Ventajas de la herencia múltiple | 155 |
| 5.5.2.2. | Inconvenientes de la herencia múltiple | 155 |
| 5.5.2.3. | Diseño de clases con herencia múltiple | 156 |
| 5.6. | Herencia repetida | 157 |
| | Resumen | 160 |
| | Ejercicios | 160 |

Parte II

PROGRAMACION ORIENTADA A OBJETOS CON C++

| | | |
|--|---|-----|
| Capítulo 6. Clases y objetos en C++ | 167 | |
| 6.1. | Clases y objetos | 168 |
| 6.2. | Objetos | 169 |
| 6.2.1. | Identificación de objetos | 170 |
| 6.3. | Clases | 171 |
| 6.4. | Creación de clases | 172 |
| 6.5. | Diagramas de clases y objetos | 173 |
| 6.6. | Construcción de clases en C++ | 176 |
| 6.6.1. | Declaración de clases | 177 |
| 6.6.2. | Definición de una clase | 179 |
| 6.6.3. | Constructores y destructores | 180 |
| 6.6.4. | Usar las clases | 181 |
| 6.6.5. | Especificación/implementación de clases | 181 |
| 6.6.6. | Compilación separada de clases | 183 |
| 6.6.7. | Reutilización de clases | 184 |
| 6.6.8. | Estilos de declaración de clases | 184 |

| | |
|---|------------|
| 6.7. Diseños prácticos de clases | 185 |
| 6.7.1. Clases <i>Reloj</i> y <i>Presentar</i> | 188 |
| 6.8. Técnicas de creación e inicialización de objetos | 190 |
| 6.8.1. Objetos dinámicos <i>new</i> y <i>delete</i> | 192 |
| 6.9. Inicialización y limpieza de objetos | 193 |
| 6.9.1. Uso de una clase | 201 |
| 6.10. Reglas prácticas para construcción de clases | 204 |
| 6.10.1. Funciones miembro | 207 |
| 6.10.2. Una aplicación sencilla | 208 |
| 6.10.3. Control de acceso a los miembros de una clase | 213 |
| 6.10.4. Creación, inicialización y destrucción de objetos | 216 |
| 6.11. El puntero <i>this</i> | 221 |
| Resumen | 222 |
| Ejercicios | 223 |
| Capítulo 7. Clases abstractas y herencia | 229 |
| 7.1. Abstracción de la generalización y especialización de clases | 230 |
| 7.2. Clases abstractas | 232 |
| 7.3. Herencia en C++: clases derivadas | 233 |
| 7.3.1. Sintaxis de la herencia simple | 233 |
| 7.3.2. Sintaxis de la herencia múltiple | 237 |
| 7.3.3. Ambigüedad y resolución de ámbito | 240 |
| 7.4. Herencia repetida y clases base virtuales | 242 |
| 7.5. Funciones virtuales puras | 243 |
| 7.5.1. Otro ejemplo de clase abstracta | 247 |
| 7.6. Diseño de clases abstractas | 247 |
| 7.7. Una aplicación práctica: jerarquía de figuras | 251 |
| 7.7.1. La clase <i>Figura</i> | 251 |
| Resumen | 252 |
| Ejercicios | 252 |
| Capítulo 8. Polimorfismo | 255 |
| 8.1. Ligadura | 256 |
| 8.1.1. Ligadura en C++ | 256 |
| 8.2. Funciones virtuales | 257 |
| 8.2.1. Ligadura dinámica mediante funciones virtuales | 258 |
| 8.3. Polimorfismo | 260 |
| 8.3.1. El polimorfismo sin ligadura dinámica | 261 |
| 8.3.2. El polimorfismo con ligadura dinámica | 262 |
| 8.4. Uso del polimorfismo | 263 |
| 8.4.1. Uso del polimorfismo en C++ | 263 |
| 8.5. Ligadura dinámica frente a ligadura estática | 264 |
| 8.6. Ventajas del polimorfismo | 265 |
| Resumen | 265 |
| Ejercicios | 266 |
| Capítulo 9. Genericidad: plantillas (<i>templates</i>) | 268 |
| 9.1. Genericidad | 269 |
| 9.2. Conceptos fundamentales de plantillas en C++ | 270 |

| | | |
|--------|--|-----|
| 9.3. | Plantillas de funciones | 271 |
| 9.3.1. | Fundamentos teóricos | 271 |
| 9.3.2. | Definición de plantilla de función | 272 |
| 9.3.3. | Un ejemplo de plantilla de funciones | 274 |
| 9.3.4. | Un ejemplo de función plantilla | 276 |
| 9.3.5. | Plantillas de función <i>ordenar</i> y <i>buscar</i> | 277 |
| 9.3.6. | Una aplicación práctica | 278 |
| 9.3.7. | Problemas en las funciones plantilla | 279 |
| 9.4. | Plantillas de clases | 280 |
| 9.4.1. | Definición de una plantilla de clase | 280 |
| 9.4.2. | Instanciación de una plantilla de clases | 283 |
| 9.4.3. | Utilización de una plantilla de clase | 283 |
| 9.4.4. | Argumentos de plantillas | 284 |
| 9.4.5. | Aplicaciones de plantillas de clases | 285 |
| 9.5. | Una plantilla para manejo de pilas de datos | 287 |
| 9.5.1. | Definición de las funciones miembro | 288 |
| 9.5.2. | Utilización de una clase plantilla | 289 |
| 9.5.3. | Instanciación de una clase plantilla con clases | 292 |
| 9.5.4. | Uso de las plantillas de funciones con clases | 292 |
| 9.6. | Plantillas frente a polimorfismo | 293 |
| | Resumen | 294 |
| | Ejercicios | 295 |

Capítulo 10. Excepciones **297**

| | | |
|---------|--|-----|
| 10.1. | Concepto de excepción | 298 |
| 10.2. | Manejo de excepciones | 298 |
| 10.2.1. | Medios típicos para el manejo de excepciones | 299 |
| 10.3. | El mecanismo de excepciones en C++ | 301 |
| 10.4. | Lanzamiento de excepciones | 302 |
| 10.4.1. | Formatos de <i>throw</i> | 303 |
| 10.5. | Manejadores de excepciones | 303 |
| 10.5.1. | Bloques <i>try</i> | 304 |
| 10.5.2. | Captura de excepciones | 305 |
| 10.5.3. | Relanzamiento de excepciones | 306 |
| 10.6. | Especificación de excepciones | 306 |
| 10.7. | Aplicaciones prácticas de manejo de excepciones | 307 |
| 10.7.1. | Calcular las raíces de una ecuación de segundo grado | 307 |
| 10.7.2. | Control de excepciones en una estructura tipo pila | 308 |
| | Resumen | 310 |
| | Ejercicios | 311 |

Capítulo 11. Reutilización de software con C++ **314**

| | | |
|---------|--|-----|
| 11.1. | Mecanismos de reutilización | 315 |
| 11.1.1. | Herramientas tradicionales de reutilización | 315 |
| 11.2. | Reutilización por herencia | 316 |
| 11.2.1. | Ventajas de la reutilización a través de la herencia | 316 |
| 11.3. | Las recompilaciones en C++ | 317 |
| 11.4. | Reutilización mediante plantillas o tipos genéricos | 318 |
| 11.4.1. | Polimorfismo frente a genericidad | 319 |

| | |
|--|-----|
| 11.5. Bibliotecas de clases | 321 |
| 11.5.1. Contenedores | 323 |
| 11.5.2. La necesidad de los contenedores | 324 |
| 11.5.3. Clases contenedoras de Borland C++ 3.1 a 5.0 | 325 |
| 11.5.4. La biblioteca estándar de plantillas (STL) | 326 |
| 11.6. Clases contenedoras en Borland C++ 4.5/5.0 | 326 |
| 11.6.1. Nombres de las clases contenedoras | 327 |
| 11.6.2. Clases vector | 328 |
| 11.6.3. Clases listas doblemente enlazadas | 328 |
| 11.6.4. Clases array | 329 |
| 11.6.5. Creación y uso de contenedores | 329 |
| Resumen | 331 |

Parte III

DISEÑO ORIENTADO A OBJETOS

Capítulo 12. Diseño orientado a objetos (Notaciones Booch, Rumbaugh y Coad/Yourdon) **335**

| | |
|---|-----|
| 12.1. Desarrollo de un sistema orientado a objetos | 336 |
| 12.1.1. Identificar clases y objetos | 337 |
| 12.1.2. Asignación de atributos y comportamiento | 338 |
| 12.1.3. Encontrar las relaciones entre clases y objetos | 340 |
| 12.1.4. Interfaz e implementación de las clases | 341 |
| 12.2. Notaciones gráficas | 342 |
| 12.2.1. Notación de Booch'93 | 344 |
| 12.2.2. Notación de Yourdon | 348 |
| 12.2.3. Notación de Rumbaugh (OMT) | 350 |
| 12.3. Implementación de clases y objetos en C++ | 353 |
| 12.3.1. El modificador <i>const</i> | 354 |
| 12.4. Creación de funciones miembro en C++ | 355 |
| 12.4.1. Funciones <i>inline</i> | 355 |
| 12.4.2. Funciones miembro virtuales y virtuales puras | 356 |
| 12.4.3. Variables miembro y accesibilidad | 357 |
| 12.5. Implementación de relaciones con C++ | 357 |
| 12.5.1. Relaciones de generalización-especialización (<i>es-un</i>) | 357 |
| 12.5.2. Relación de agregación/composición (<i>tiene-un</i>) | 362 |
| 12.5.3. Relación de asociación | 366 |
| 12.5.4. Relación <i>utiliza (uses)</i> | 367 |
| 12.6. Clases abstractas | 368 |
| 12.6.1. Abstracción mediante plantillas | 372 |
| 12.7. Una aplicación orientada a objetos | 372 |
| 12.7.1. Identificar las clases | 373 |
| 12.7.2. Identificar relaciones | 373 |
| 12.7.3. Definir el interfaz de cada clase | 376 |
| Resumen | 379 |
| Ejercicios | 381 |

Parte IV
EL LENGUAJE C++: SINTAXIS, CONSTRUCCION
Y PUESTA A PUNTO DE PROGRAMAS

| | |
|---|------------|
| Capítulo 13. De C a C++ | 385 |
| 13.1. Limitaciones de C | 386 |
| 13.2. Mejora de características de C en C++ | 386 |
| 13.3. El primer programa C++ | 388 |
| 13.3.1. Comparación de programas C y C++ | 389 |
| 13.4. Nuevas palabras reservadas de C++ | 390 |
| 13.5. Comentarios | 391 |
| 13.6. Declaraciones de variables | 392 |
| 13.6.1. Declaración de variables en un bucle <i>for</i> | 394 |
| 13.6.2. Declaraciones externas | 395 |
| 13.6.3. El ámbito de una variable | 396 |
| 13.7. El especificador de tipos <i>const</i> | 398 |
| 13.7.1. Diferencias entre <i>const</i> de C++ y <i>const</i> de C | 400 |
| 13.7.2. Las variables volátiles | 402 |
| 13.8. Especificador de tipo <i>void</i> | 403 |
| 13.8.1. Punteros <i>void</i> | 403 |
| 13.9. Los tipos <i>char</i> | 404 |
| 13.9.1. Inicialización de caracteres | 404 |
| 13.10. Cadenas | 405 |
| 13.11. Conversión obligatoria de tipos (<i>Casting</i>) | 406 |
| 13.12. El especificador de tipo <i>volatile</i> | 407 |
| 13.13. Estructuras, uniones y enumeraciones | 408 |
| 13.13.1. Estructuras y uniones | 409 |
| 13.13.2. Uniones anónimas | 410 |
| 13.13.3. Enumeraciones | 411 |
| 13.13.4. Enumeraciones anónimas | 413 |
| 13.14. Funciones en C++ | 414 |
| 13.14.1. <i>main()</i> | 414 |
| 13.14.2. Prototipos de funciones | 415 |
| 13.14.3. Una declaración típica de funciones y prototipos | 417 |
| 13.14.4. Funciones en línea | 419 |
| 13.14.5. Ventajas sobre las macros | 421 |
| 13.14.6. Argumentos por omisión | 422 |
| 13.14.7. Funciones con un número variable de parámetros (el parámetro...) | 425 |
| 13.15. Llamada a funciones C. Programas mixtos C/C++ | 426 |
| 13.16. El tipo referencia | 427 |
| 13.17. Sobrecarga | 432 |
| 13.17.1. Sobrecarga de funciones | 432 |
| 13.17.2. Aplicación de sobrecarga de funciones | 435 |
| 13.17.3. Sobrecarga de operadores | 437 |
| 13.18. Asignación dinámica de memoria | 438 |
| 13.18.1. El operador <i>new</i> | 439 |
| 13.18.2. El puntero nulo/cero | 441 |
| 13.18.3. El operador <i>delete</i> | 441 |
| 13.18.4. Ventajas de <i>new</i> y <i>delete</i> | 442 |

| | |
|---|-----|
| 13.19. Organización de un programa C++ | 443 |
| 13.19.1. Evitar definiciones múltiples | 444 |
| 13.19.2. Evitar incluir archivos de cabecera más de una vez | 445 |
| Resumen | 447 |
| Ejercicios | 448 |

Capítulo 14. Construcción de programas en C++/C 450

| | |
|--|-----|
| 14.1. Compilación separada de programas | 451 |
| 14.1.1. Programas multiarchivo | 452 |
| 14.1.2. Bibliotecas de clases | 452 |
| 14.2. Almacenamiento <i>extern</i> y <i>static</i> | 453 |
| 14.2.1. <i>extern</i> | 453 |
| 14.2.2. <i>static</i> | 455 |
| 14.3. Estructura de un programa C | 456 |
| 14.4. Compilación separada de clases | 458 |
| 14.5. Estructura de un programa C++ | 460 |
| 14.5.1. ¿Qué son archivos de cabecera? | 462 |
| 14.5.2. Inclusión de archivos | 463 |
| 14.6. Programas multiarchivo | 463 |
| 14.6.1. ¿Qué se debe poner en un archivo fuente? | 464 |
| 14.6.2. Referencias externas | 465 |
| 14.7. Construcción de archivos proyecto | 466 |
| 14.7.1. Abrir un proyecto | 467 |
| 14.7.2. Añadir archivos fuente | 467 |
| 14.8. Transporte de aplicaciones desde C a C++ | 468 |
| 14.8.1. Enlace entre programas C y C++ | 468 |
| Resumen | 470 |
| Ejercicios | 471 |

Capítulo 15. Puesta a punto de programas en C++. Errores de programación típicos 474

| | |
|---|-----|
| 15.1. Depuración de programas | 475 |
| 15.1.1. Errores durante la depuración | 476 |
| 15.2. Errores en arrays | 476 |
| 15.3. Errores en cadenas | 477 |
| 15.3.1. Cálculo incorrecto de la longitud de una cadena | 478 |
| 15.4. Errores en comentarios | 479 |
| 15.5. Errores en corchetes y llaves | 480 |
| 15.6. Errores en funciones | 480 |
| 15.6.1. Pasar un argumento por valor en lugar de por variable | 481 |
| 15.6.2. Fallos en el valor de retorno de la función | 481 |
| 15.6.3. No incluir el archivo de cabecera de una función en tiempo de ejecución | 482 |
| 15.7. Errores en macros | 482 |
| 15.7.1. Omisión de paréntesis en los argumentos de macros | 483 |
| 15.7.2. Especificación no válida de macros tipo función | 483 |
| 15.8. Errores con operadores | 484 |
| 15.8.1. Mal uso de operadores de incremento (++) y decremento (--). | 484 |
| 15.8.2. Confusión de operadores de asignación | 484 |
| 15.8.3. Fallos en la precedencia de operadores | 485 |