1.7.2. Modelado del mundo real

Otro problema importante de la programación estructurada reside en el hecho de que la disposición separada de datos y funciones no se corresponden con los modelos de las cosas del mundo real. En el mundo físico se trata con objetos físicos tales como personas, autos o aviones. Estos objetos no son como los datos ni como las funciones. Los objetos complejos o no del mundo real tienen *atributos* y *comportamiento.*

Los **atributos** o características de los objetos son, por ejemplo: en las personas, su edad, su profesión, su domicilio, etc.; en un auto, la potencia, el número de matrícula, el precio, número de puertas, etc.; en una casa, la superficie, el precio, el año de construcción, la dirección, etcétera. En realidad, los atributos del mundo real tienen su equivalente en los datos de un programa; toman un valor específico, tal como 200 metros cuadrados, 20.000 dólares, cinco puertas, etc.

El **comportamiento** son las acciones que ejecutan los objetos del mundo real como respuesta a un determinado estímulo. Si usted pisa los frenos en un auto, el coche (carro) se detiene; si acelera, el auto aumenta su velocidad, etc. El comportamiento, en esencia, es como una función: se llama a una función para hacer algo (visualizar la nómina de los empleados de una empresa).

Por estas razones, ni los datos ni las funciones, por sí mismas, modelan los objetos del mundo real de un modo eficiente.

> La programación estructurada mejora la claridad, fiabilidad y facilidad de mantenimiento de los programas; sin embargo, para programas grandes o a gran escala, presentan retos de difícil solución.

1.8. PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos, tal vez el paradigma de programación más utilizado en el mundo del desarrollo de *software* y de la ingeniería de *software* del siglo XXI, trae un nuevo enfoque a los retos que se plantean en la programación estructurada cuando los problemas a resolver son complejos. Al contrario que la programación *procedimental* que enfatiza en los algoritmos, la POO enfatiza en los datos. En lugar de intentar ajustar un problema al enfoque *procedimental* de un lenguaje, POO intenta ajustar el lenguaje al problema. La idea es diseñar formatos de datos que se correspondan con las características esenciales de un problema.

La idea fundamental de los lenguajes orientados a objetos es combinar en una única unidad o módulo, tanto los datos como las funciones que operan sobre esos datos. Tal unidad se llama **objeto**.

Las funciones de un objeto se llaman *funciones miembro* en C++ o *métodos* (éste es el caso de Smalltalk, uno de los primeros lenguajes orientados a objetos), y son el único medio para acceder a sus datos. Los datos de un objeto, se conocen también como *atributos* o *variables de instancia.* Si se desea leer datos de un objeto, se llama a una función miembro del objeto. Se accede a los datos y se devuelve un valor. No se puede acceder a los datos directamente. Los datos están ocultos, de modo que están protegidos de alteraciones accidentales. Los datos y las funciones se dice que están *encapsulados en una única entidad. El encapsulamiento de datos y la ocultación* de los datos son términos clave en la descripción de lenguajes orientados a objetos.

Si se desea modificar los datos de un objeto, se conoce exactamente cuáles son las funciones que interactúan con las funciones miembro del objeto. Ninguna otra función puede acceder a los datos. Esto simplifica la escritura, depuración y mantenimiento del programa. Un programa C++ se compone, normalmente, de un número de objetos que se comunican unos con otros mediante la llamada a otras funciones miembro. La organización de un programa en C++ se muestra en la Figura 1.5. La llamada a una función miembro de un objeto se denomina *enviar un mensaje* a otro objeto.

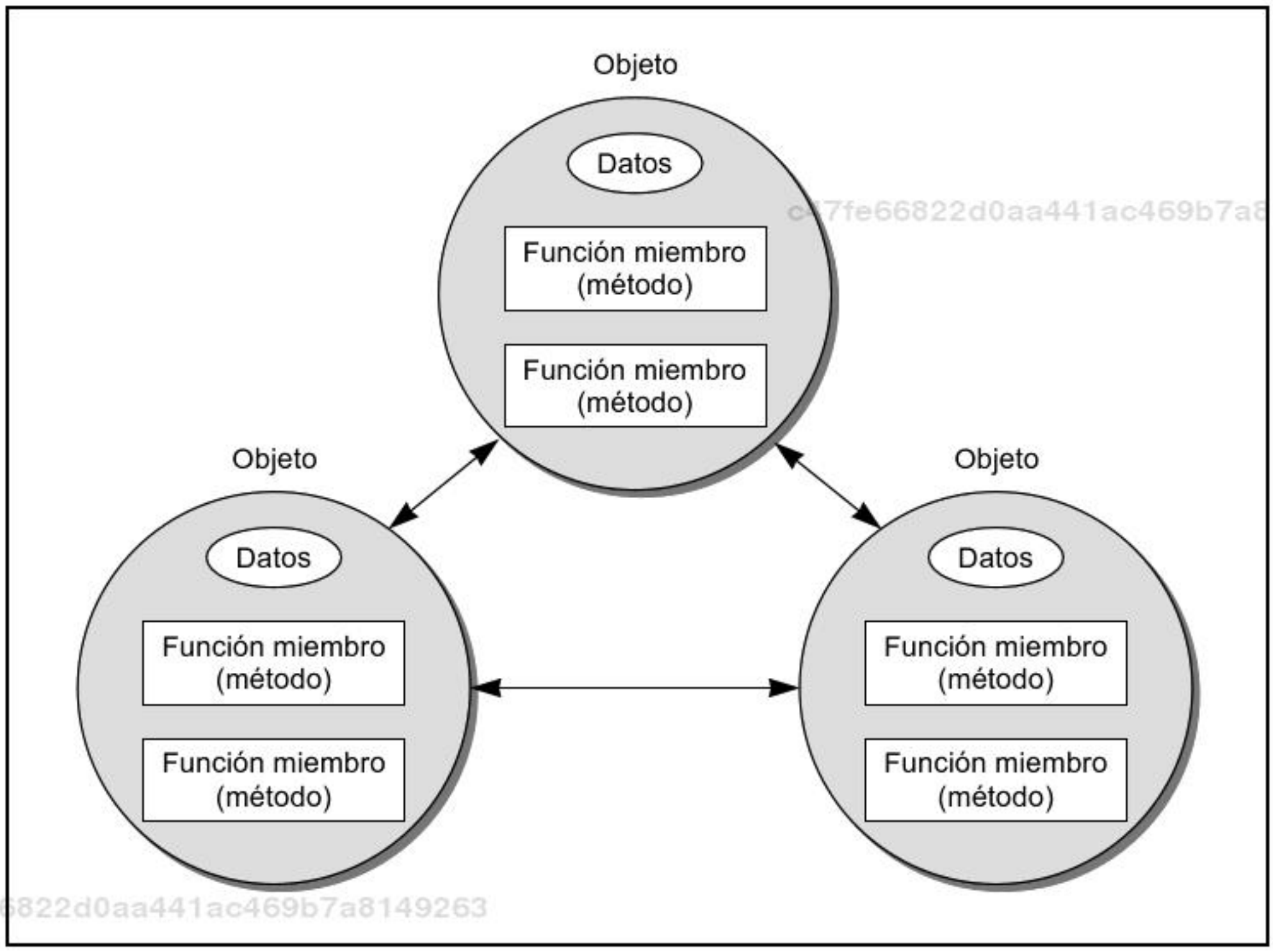


Figura 1.5. Organización típica de un programa orientado a objetos.

En el paradigma orientado a objetos, el programa se organiza como un conjunto finito de objetos que contiene datos y operaciones (funciones miembro en C++) que llaman a esos datos y que se comunican entre sí mediante mensajes.

1.8.1. Propiedades fundamentales de la orientación a objetos

Existen diversas características ligadas a la orientación a objetos. Todas las propiedades que se suelen considerar, no son exclusivas de este paradigma, ya que pueden existir en otros paradigmas, pero en su conjunto definen claramente los lenguajes orientados a objetos. Estas propiedades son:

- *Abstracción* (tipos abstractos de datos y clases).
- *Encapsulado* o *encapsulamiento* de datos.
- *Ocultación* de datos.
- *Herencia*.
- *Polimorfismo*.

C++ soporta todas las características anteriores que definen la orientación a objetos, aunque hay numerosas discusiones en torno a la consideración de C++ como lenguaje orientado a objetos. La razón es que, en contraste con lenguajes tales como Smalltalk, Java o C#, C++ no es un lenguaje orientado a objetos puro. C++ soporta orientación a objetos pero es compatible con C y permite que programas C++ se escriban sin utilizar características orientadas a objetos. De hecho, C++ es un lenguaje *multiparadigma* que permite *programación estructurada*, *procedimental*, *orientada a objetos* y *genérica*.

1.8.2. Abstracción

La abstracción es la propiedad de los objetos que consiste en tener en cuenta sólo los aspectos más importantes desde un punto de vista determinado y no tener en cuenta los restantes aspectos. El término **abstracción**, que se suele utilizar en programación, se refiere al hecho de diferenciar entre las propiedades externas de una entidad y los detalles de la composición interna de dicha entidad. Es la abstracción la propiedad que permite ignorar los detalles internos de un dispositivo complejo tal como una computadora, un automóvil, una lavadora o un horno de microondas, etc., y usarlo como una única unidad comprensible. Mediante la abstracción se diseñan y fabrican estos sistemas complejos en primer lugar y, posteriormente, los componentes más pequeños de los cuales están compuestos. Cada componente representa un nivel de abstracción en el cual el uso del componente se aísla de los detalles de la composición interna del componente. La abstracción posee diversos grados denominados niveles de abstracción.

En consecuencia, la abstracción posee diversos grados de complejidad que se denominan *niveles de abstracción* que ayudan a estructurar la complejidad intrínseca que poseen los sistemas del mundo real. En el modelado orientado a objetos de un sistema esto significa centrarse en *qué* es *y qué hace* un objeto y no en *cómo* debe implementarse. Durante el proceso de abstracción es cuando se decide qué características y comportamiento debe tener el modelo.

Aplicando la abstracción se es capaz de construir, analizar y gestionar sistemas de computadoras complejos y grandes que no se podrían diseñar si se tratara de modelar a un nivel detallado. En cada nivel de abstracción se visualiza el sistema en términos de componentes, denominados **herramientas abstractas**, cuya composición interna se ignora. Esto nos permite concentrarnos en cómo cada componente interactúa con otros componentes y centrarnos en la parte del sistema que es más relevante para la tarea a realizar en lugar de perderse a nivel de detalles menos significativos.

En estructuras o registros, las propiedades individuales de los objetos se pueden almacenar en los miembros. Para los objetos es de interés *cómo* están organizados sino también *qué* se puede hacer con ellos. Es decir, las operaciones que forman la composición interna de un objeto son también importantes. El primer concepto en el mundo de la orientación a objetos nació con los tipos abstractos de datos (**TAD**). Un tipo abstracto de datos describe no sólo los atributos de un objeto, sino también su comportamiento (las operaciones). Esto puede incluir también una descripción de los estados que puede alcanzar un objeto.

Un medio de reducir la complejidad es la abstracción. Las características y los procesos se reducen a las propiedades esenciales, son resumidas o combinadas entre sí. De este modo, las características complejas se hacen más manejables.

EJEMPLO 1.7. Diferentes modelos de abstracción del término coche (carro).

- Un `coche (carro)` es la combinación (o composición) de diferentes partes, tales como motor, carrocería, cuatro ruedas, cinco puertas, etc.
- Un `coche (carro)` es un concepto común para diferentes tipos de coches. Pueden clasificarse por el nombre del fabricante (Audi, BMW, Seat, Toyota, Chrisler...), por su categoría (turismo, deportivo, todoterreno...), por el carburante que utilizan (gasolina, gasoil, gas, híbrido...).

La abstracción `coche` se utilizará siempre que la marca, la categoría o el carburante no sean significativos. Así, un `carro (coche)` se utilizará para transportar personas o ir de `Carchelejo` a `Cazorla`.

1.8.3. Encapsulación y ocultación de datos

El *encapsulado* o *encapsulación de datos* es el proceso de agrupar datos y operaciones relacionadas bajo la misma unidad de programación. En el caso de los objetos que poseen las mismas características y comportamiento se agrupan en clases, que no son más que unidades o módulos de programación que encapsulan datos y operaciones.

La ocultación de datos permite separar el aspecto de un componente, definido por su *interfaz* con el exterior, de sus detalles internos de implementación. Los términos **ocultación de la información** (*information hiding*) y **encapsulación de datos** (*data encapsulation*) se suelen utilizar como sinónimos, pero no siempre es así, a veces se utilizan en contextos diferentes. En C++ no es lo mismo, los datos internos están protegidos del exterior y no se pueden acceder a ellos más que desde su propio interior y, por tanto, no están ocultos. El acceso al objeto está restringido sólo a través de una interfaz bien definida.

El diseño de un programa orientado a objetos contiene, al menos, los siguientes pasos:

1. Identificar los *objetos* del sistema.
2. Agrupar en *clases* a todos los objetos que tengan características y comportamiento comunes.
3. Identificar los *datos* y *operaciones* de cada una de las clases.
4. Identificar las *relaciones* que pueden existir entre las clases.

En C++, un **objeto** es un elemento individual con su propia identidad; por ejemplo, un libro, un automóvil... Una **clase** puede describir las propiedades genéricas de un ejecutivo de una empresa (`nombre`, `título`, `salario`, `cargo`...) mientras que un objeto representará a un ejecutivo específico (`Luis Mackoy`, director general). En general, una clase define qué datos se utilizan para representar un objeto y las operaciones que se pueden ejecutar sobre esos datos.

Cada clase tiene sus propias características y comportamiento; en general, una clase define los datos que se utilizan y las operaciones que se pueden ejecutar sobre esos datos. Una clase describe un conjunto de objetos. En el sentido estricto de programación, una clase es un tipo de datos. Diferentes variables se pueden crear de este tipo. En programación orientada a objetos, éstas se llaman *instancias*. Las instancias son, por consiguiente, la realización de los objetos descritos en una clase. Estas instancias constan de datos o atributos descritos en la clase y se pueden manipular con las operaciones definidas dentro de ellas.

Los términos *objeto* e *instancia* se utilizan frecuentemente como sinónimos (especialmente en C++). Si una variable de tipo `Carro` se declara, se crea un objeto `Carro` (una instancia de la clase `Carro`).

Las operaciones definidas en los objetos se llaman *métodos*. Cada operación llamada por un objeto se interpreta como un *mensaje* al objeto, que utiliza un método específico para procesar la operación.

En el diseño de programas orientados a objetos se realiza en primer lugar el diseño de las clases que representan con precisión aquellas cosas que trata el programa. Por ejemplo, un programa de dibujo, puede definir clases que representan rectángulos, líneas, pinceles, colores, etcétera. Las definiciones de clases, incluyen una descripción de operaciones permisibles para cada clase, tales como desplazamiento de un círculo o rotación de una línea. A continuación se prosigue el diseño de un programa utilizando objetos de las clases.

El diseño de clases fiables y útiles puede ser una tarea difícil. Afortunadamente, los lenguajes POO facilitan la tarea ya que incorporan clases existentes en su propia programación. Los fabricantes de *software* proporcionan numerosas bibliotecas de clases, incluyendo bibliotecas de clases diseñadas para simplificar la creación de programas para entornos tales como Windows, Linux, Macintosh, Unix o Vista. Uno de los beneficios reales de C++ es que permite la reutilización y adaptación de códigos existentes y ya bien probados y depurados.

1.8.4. Objetos

El objeto es el centro de la programación orientada a objetos. Un **objeto** es algo que se visualiza, se utiliza y juega un rol o papel. Si se programa con enfoque orientado a objetos, se intenta descubrir e implementar los objetos que juegan un rol en el dominio del problema y en consecuencia del programa. La estructura interna y el comportamiento de un objeto, en una primera fase, no tiene prioridad. Es importante que un objeto tal como un carro o una casa juegan un rol.

Dependiendo del problema, diferentes aspectos de un dominio son relevantes. Un carro puede ser ensamblado por partes tales como un motor, una carrocería, unas puertas o puede ser descrito utilizando propiedades tales como su velocidad, su kilometraje o su fabricante. Estos atributos indican el objeto. De modo similar una persona, también se puede ver como un objeto, del cual se disponen diferentes atributos. Dependiendo de la definición del problema, esos atributos pueden ser el nombre, apellido, dirección, número de teléfono, color del cabello, altura, peso, profesión, etc.

Un objeto no necesariamente ha de realizar algo concreto o tangible. Puede ser totalmente abstracto y también puede describir un proceso. Por ejemplo, un partido de baloncesto o de *rugby* puede ser descrito como un objeto. Los atributos de este objeto pueden ser los jugadores, el entrenador, la puntuación y el tiempo transcurrido de partido.

Cuando se trata de resolver un problema con orientación a objetos, dicho problema no se descompone en funciones como en programación estructurada tradicional, caso de C, sino en objetos. El pensar en términos de objetos tiene una gran ventaja: se asocian los objetos del problema a los objetos del mundo real.

¿Qué tipos de cosas son objetos en los programas orientados a objetos? La respuesta está limitada por su imaginación aunque se pueden agrupar en categorías típicas que facilitarán su búsqueda en la definición del problema de un modo más rápido y sencillo.

- Recursos Humanos:
  - — Empleados.
  - — Estudiantes.
  - — Clientes.

— Vendedores.
— Socios.

• Colecciones de datos:

— Arrays (arreglos).
— Listas.
— Pilas.
— Árboles.
— Árboles binarios.
— Grafos.

• Tipos de datos definidos por usuarios:

— Hora.
— Números complejos.
— Puntos del plano.
— Puntos del espacio.
— Ángulos.
— Lados.

• Elementos de computadoras:

— Menús.
— Ventanas.
— Objetos gráficos (rectángulos, círculos, rectas, puntos…).
— Ratón (mouse).
— Teclado.
— Impresora.
— USB.
— Tarjetas de memoria de cámaras fotográficas.

• Objetos físicos:

— Carros.
— Aviones.
— Trenes.
— Barcos.
— Motocicletas.
— Casas.

• Componentes de videojuegos:

— Consola.
— Mandos.
— Volante.
— Conectores.
— Memoria.
— Acceso a Internet.

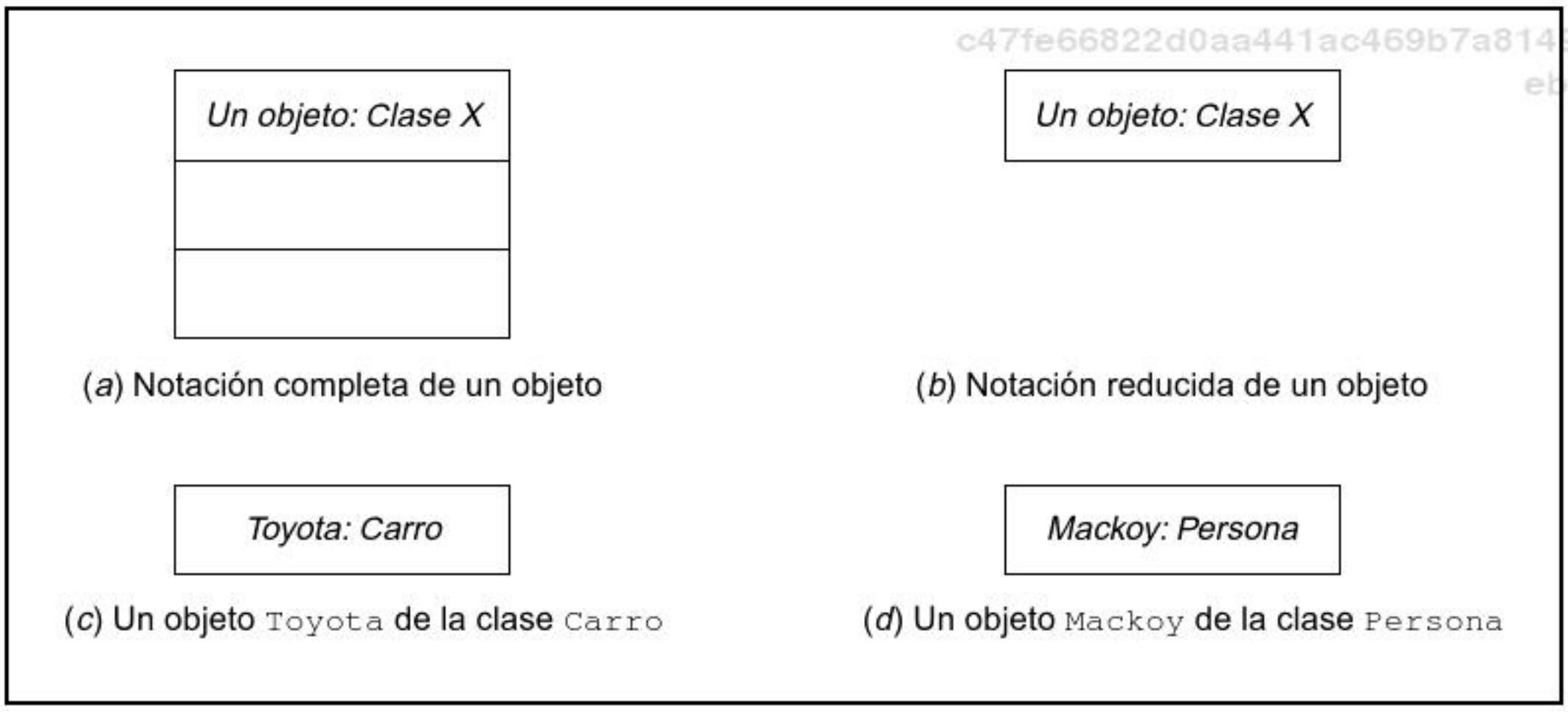
La correspondencia entre objetos de programación y objetos del mundo real es el resultado eficiente de combinar datos y funciones que manipulan esos datos. Los objetos resultantes

ofrecen una mejor solución al diseño del programa que en el caso de los lenguajes orientados a procedimientos.

Un **objeto** se puede definir desde el punto de vista conceptual como una entidad individual de un sistema y que se caracteriza por un estado y un comportamiento. Desde el punto de vista de implementación, un **objeto** es una entidad que posee un conjunto de *datos* y un conjunto de *operaciones* (*funciones* o *métodos*).

El estado de un objeto viene determinado por los valores que toman sus datos, cuyos valores pueden tener las restricciones impuestas en la definición del problema. Los datos se denominan también *atributos* y componen la estructura del objeto y las operaciones —también llamadas *métodos*— representan los servicios que proporciona el objeto.

La representación gráfica de un objeto en **UML** se muestra en la Figura 1.6.



(*a*) Notación completa de un objeto

(*b*) Notación reducida de un objeto

(*c*) Un objeto `Toyota` de la clase `Carro`

(*d*) Un objeto `Mackoy` de la clase `Persona`

Figura 1.6. Representación de objetos en UML (Lenguaje Unificado de Modelado).

1.8.5. Clases

En POO los objetos son miembros o instancias de clases. En esencia, una **clase** es un tipo de datos al igual que cualquier otro tipo de dato definido en un lenguaje de programación. La diferencia reside en que la clase es un tipo de dato que contiene datos y funciones. Una clase describe muchos objetos y es preciso definirla, aunque su definición no implica creación de objetos (Figura 1.7).

Una **clase** es, por consiguiente, una descripción de un número de objetos similares. `Madonna`, `Sting`, `Prince`, `Juanes`, `Carlos Vives` o `Juan Luis Guerra` son miembros u objetos de la clase `"músicos de rock"`. Un objeto concreto, `Juanes` o `Carlos Vives`, son *instancias* de la clase `"músicos de rock"`.

En **C++** una clase es una estructura de dato o tipo de dato que contiene funciones (métodos) como miembros y datos. Una clase es una descripción general de un conjunto de objetos similares. Por definición, todos los objetos de una clase comparten los mismos atributos (datos) y las mismas operaciones (métodos). Una clase encapsula las abstracciones de datos y operaciones necesarias para describir una entidad u objeto del mundo real.

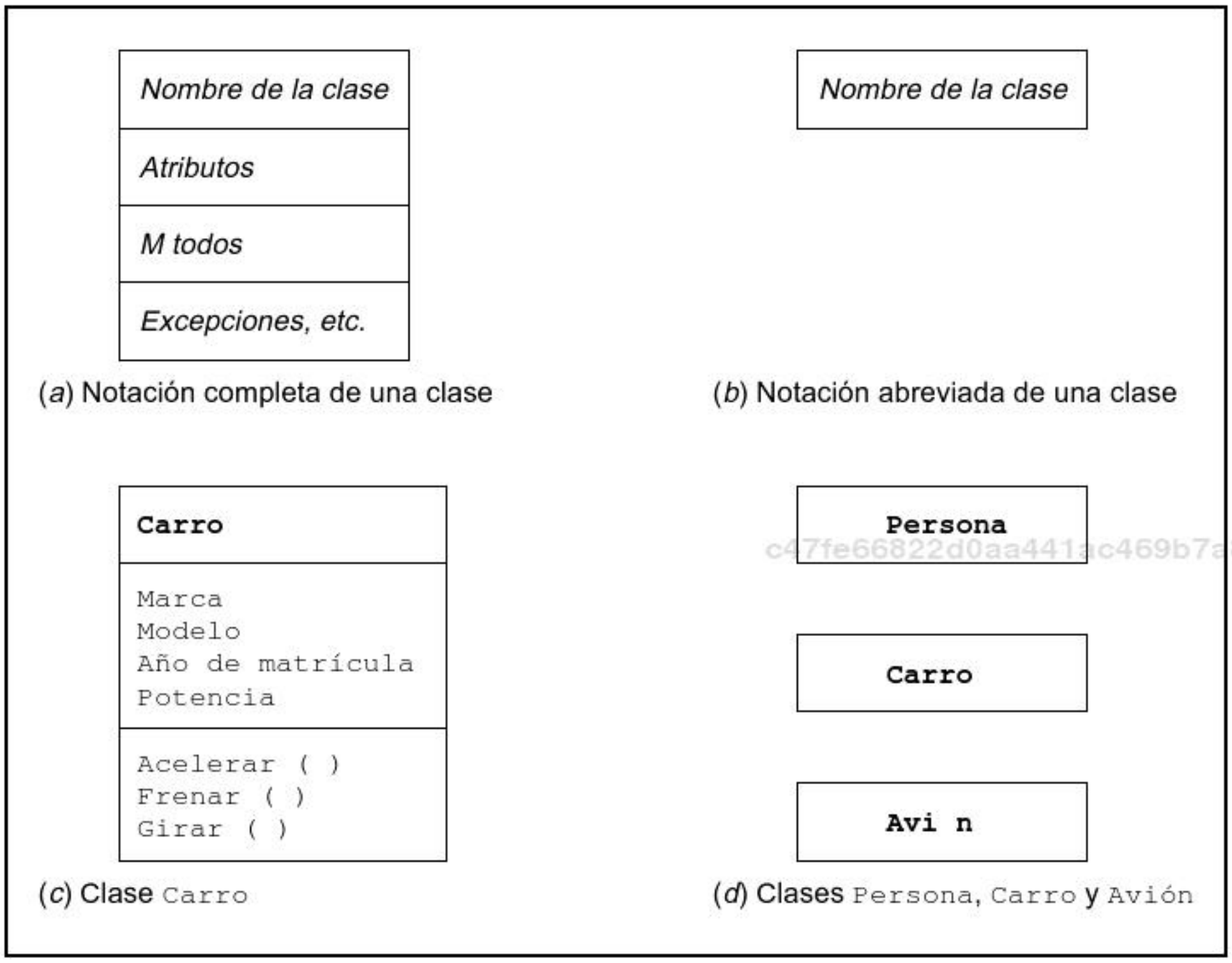


Figura 1.7. Representación de clases en UML.

Una clase se representa en **UML** mediante un rectángulo que contiene en una banda con el nombre de la clase y opcionalmente otras dos bandas con el nombre de sus atributos y de sus operaciones o métodos (Figura 1.8).

Figura 1.8. Representación de clases en **UML** con atributos y métodos.

1.8.6. Generalización y especialización: herencia

La *generalización* es la propiedad que permite compartir información entre dos entidades evitando la redundancia. En el comportamiento de objetos existen con frecuencia propiedades que son comunes en diferentes objetos y esta propiedad se denomina *generalización*.

Por ejemplo, máquinas lavadoras, frigoríficos, hornos de microondas, tostadoras, lavavajillas, etc., son todos electrodomésticos (aparatos del hogar). En el mundo de la orientación a objetos, cada uno de estos aparatos es un **subclase** de la clase `Electrodoméstico` y a su vez `Electrodoméstico` es una **superclase** de todas las otras clases `(máquinas lavadoras, frigoríficos, hornos de microondas, tostadoras, lavavajillas,...)`. El proceso inverso de la generalización por el cual se definen nuevas clases a partir de otras ya existentes se denomina *especialización*.

En orientación a objetos, el mecanismo que implementa la propiedad de generalización se denomina **herencia**. La herencia permite definir nuevas clases a partir de otras clases ya existentes, de modo que presentan las mismas características y comportamiento de éstas, así como otras adicionales.

La idea de clases conduce a la idea de herencia. Clases diferentes se pueden conectar unas con otras de modo jerárquico. Como ya se ha comentado anteriormente con las relaciones de generalización y especialización, en nuestras vidas diarias se utiliza el concepto de clases divididas en subclases. La clase `Animal` se divide en `Anfibios`, `Mamíferos`, `Insectos`, `Pájaros`, etc., y la clase `Vehículo` en `Carros`, `Motos`, `Camiones`, `Buses`, etc.

El principio de la división o clasificación es que cada subclase comparte características comunes con la clase de la que procede o se deriva. Los carros, motos, camiones y buses tienen ruedas, motores y carrocerías; son las características que definen a un vehículo. Además de las características comunes con los otros miembros de la clase, cada subclase tiene sus propias características. Por ejemplo, los camiones tienen una cabina independiente de la caja que transporta la carga; los buses tienen un gran número de asientos independientes para los viajeros que ha de transportar, etc. En la Figura 1.9 se muestran clases pertenecientes a una jerarquía o herencia de clases.

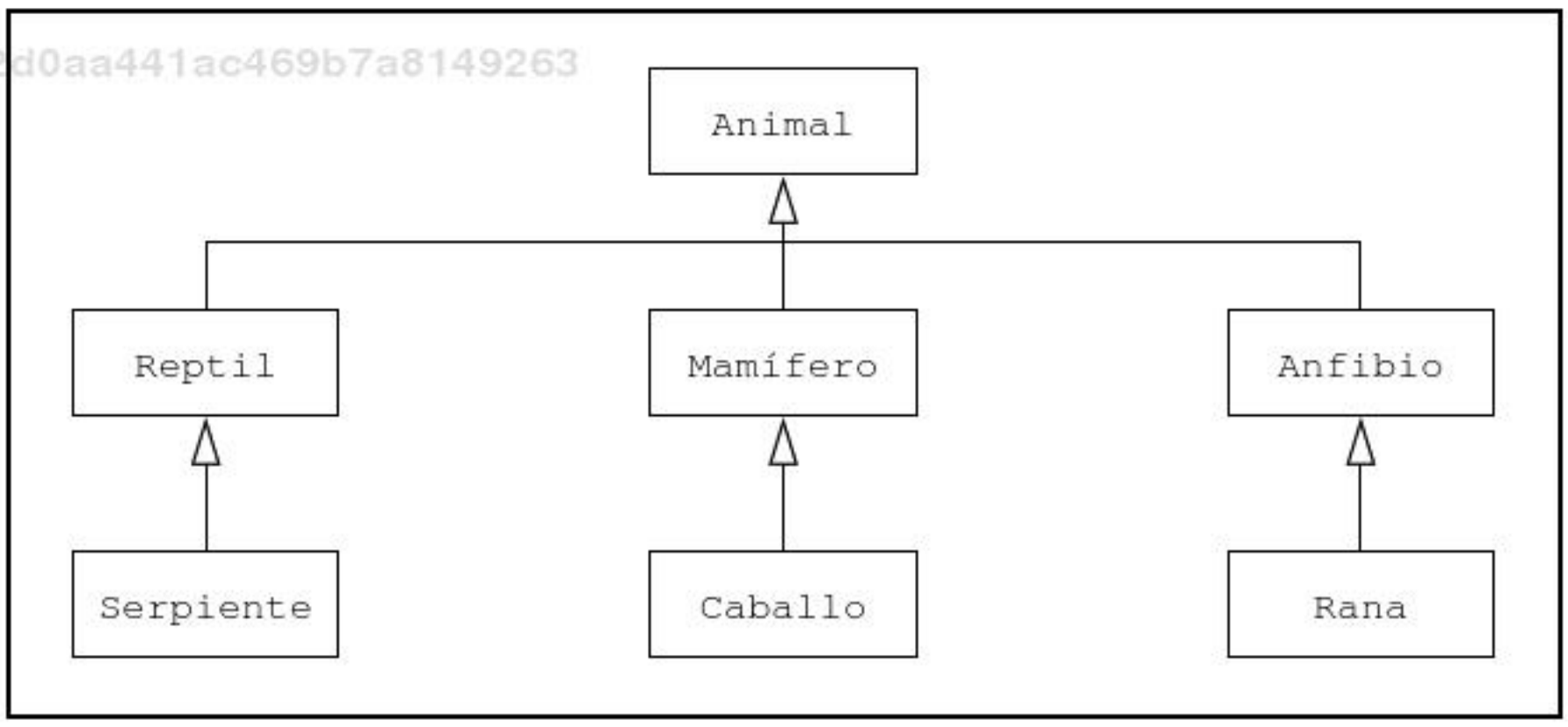


Figura 1.9. Herencia de clases en UML.

De modo similar una clase se puede convertir en padre o raíz de otras subclases. En C++ la clase original se denomina *clase base* y las clases que se derivan de ella se denominan *clases*

derivadas y siempre son una especialización o *concreción* de su clase base. A la inversa, la clase base es la generalización de la clase derivada. Esto significa que todas las propiedades (atributos y operaciones) de la clase base se heredan por la clase derivada, normalmente suplementada con propiedades adicionales.

1.8.7. Reusabilidad

Una vez que una clase ha sido escrita, creada y depurada, se puede distribuir a otros programadores para utilizar en sus propios programas. Esta propiedad se llama *reusabilidad*[9] o *reutilización*. Su concepto es similar a las funciones incluidas en las bibliotecas de funciones de un lenguaje procedimental como C que se pueden incorporar en diferentes programas.

En C++, el concepto de herencia proporciona una extensión o ampliación al concepto de *reusabilidad*. Un programador puede considerar una clase existente y sin modificarla, añadir competencias y propiedades adicionales a ella. Esto se consigue derivando una nueva clase de una ya existente. La nueva clase heredará las características de la clase antigua, pero es libre de añadir nuevas características propias.

La facilidad de reutilizar o reusar el *software* existente es uno de los grandes beneficios de la POO: muchas empresas consiguen con la reutilización de clases en nuevos proyectos la reducción de los costes de inversión en sus presupuestos de programación. ¿En esencia cuáles son las ventajas de la herencia? Primero, se utiliza para consistencia y reducir código, propiedades comunes de varias clases sólo necesitan ser implementadas una vez y sólo necesitan modificarse una vez si es necesario. Tercero, el concepto de abstracción de la funcionalidad común está soportada.

1.8.8. Polimorfismo

Además de las ventajas de consistencia y reducción de código, la herencia, aporta también otra gran ventaja: facilitar el polimorfismo. **Polimorfismo** es la propiedad de que un operador o una función actúen de modo diferente en función del objeto sobre el que se aplican. En la practica, el polimorfismo significa la capacidad de una operación de ser interpretada sólo por el propio objeto que lo invoca. Desde un punto de vista práctico de ejecución del programa, el polimorfismo se realiza en tiempo de ejecución ya que durante la compilación no se conoce qué tipo de objeto y por consiguiente qué operación ha sido llamada.

La propiedad de **polimorfismo** es aquella en que una operación tiene el mismo nombre en diferentes clases, pero se ejecuta de diferentes formas en cada clase. Así, por ejemplo, la operación de `abrir` se puede dar en diferentes clases: abrir una puerta, abrir una ventana, abrir un periódico, abrir un archivo, abrir una cuenta corriente en un banco, abrir un libro, etc. En cada caso se ejecuta una operación diferente aunque tiene el mismo nombre en todos ellos `"abrir"`. El polimorfismo es la propiedad de una operación de ser interpretada sólo por el objeto al que pertenece. Existen diferentes formas de implementar el polimorfismo y variará dependiendo del lenguaje de programación. Veamos el concepto con ejemplos de la vida diaria.

En un taller de reparaciones de automóviles existen numerosos carros (coches), de marcas diferentes, de modelos diferentes, de tipos diferentes, potencias diferentes, etc. Constituyen una

---

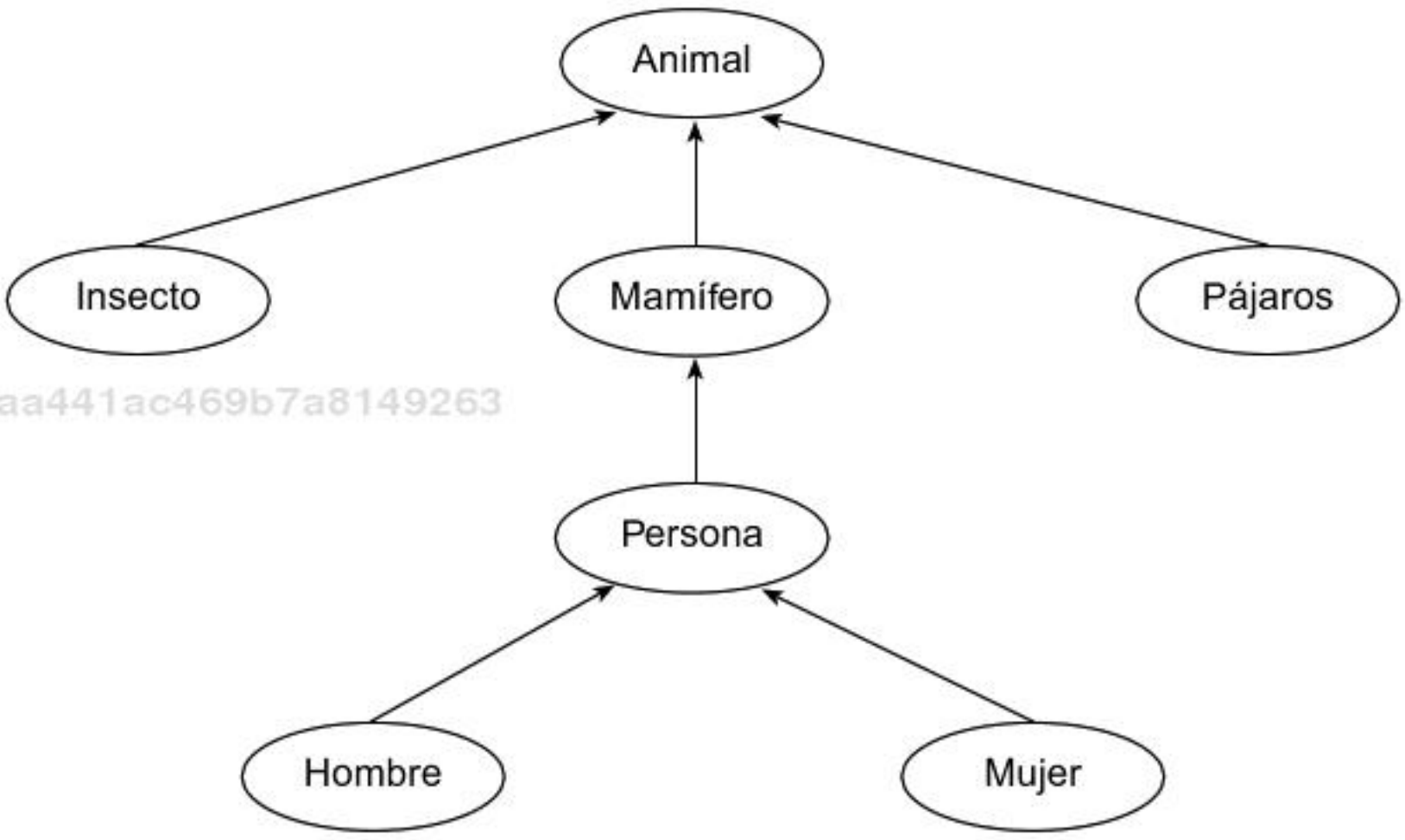
[9] El término proviene del concepto ingles *reusability*. La traducción no ha sido aprobada por la RAE, pero se incorpora al texto por su gran uso y difusión entre los profesionales de la informática.

clase o colección heterogénea de carros (coches). Supongamos que se ha de realizar una operación común `"cambiar los frenos del carro"`. La operación a realizar es la misma, incluye los mismos principios; sin embargo, dependiendo del coche, en particular, la operación será muy diferente, incluirá diferentes acciones en cada caso. Otro ejemplo a considerar y relativo a los operadores "+" y "*" aplicados a números enteros o números complejos; aunque ambos son números, en un caso la suma y multiplicación son operaciones simples, mientras que en el caso de los números complejos al componerse de parte real y parte imaginaria, será necesario seguir un método específico para tratar ambas partes y obtener un resultado que también será un número complejo.

El uso de operadores o funciones de forma diferente, dependiendo de los objetos sobre los que están actuando se llama polimorfismo (una cosa con diferentes formas). Sin embargo, cuando un operador existente, tal como + o =, se le permite la posibilidad de operar sobre nuevos tipos de datos, se dice entonces que el operador está sobrecargado. La sobrecarga es un tipo de polimorfismo y una característica importante de la POO.

EJEMPLO 1.8. Definir una jerarquía de clases para: animal, insecto, mamíferos, pájaros, persona hombre y mujer. Realizar un definición en pseudocódigo de las clases.

Las clases de objeto `Mamífero`, `Pájaro` e `Insecto` se definen como *subclases* de `Animal`; la clase de objeto `Persona`, como una subclase de `Mamífero`, y un `Hombre` y una `Mujer` son subclases de `Persona`.



Las definiciones de clases para esta jerarquía puede tomar la siguiente estructura:

```
clase Animal
  atributos (propiedades)
    string: tipo;
    real: peso;
    (...algun tipo de habitat...):habitat;
  operaciones
```

```
        crear()→ criatura;
        predadores(criatura)→ fijar(criatura);
        esperanza_vida(criatura) → entero;
        ...
    fin criatura

    clase Insecto hereda Animal
        atributos (propiedades)
          cadena: nombre
        operaciones
        ...
    fin Insecto

    clase Mamifero hereda Animal;
        atributos (propiedades)
          real: periodo_gestacion;
        operaciones
          ...
    fin Mamifero

    clase Pajaro hereda Animal
        atributos (propiedades)
          cadena: nombre
          entero: Alas
        operaciones
        ...
    fin Pajaro

    clase Persona hereda Mamifero;
        atributos (propiedades)
          string: apellido, nombre;
          date: fecha_nacimiento;
          pais: origen;
    fin Persona

    clase Hombre hereda Persona;
        atributos (propiedades)
          mujer: esposa;
          ...
        operaciones
          ...
    fin Hombre

    clase Mujer hereda Persona;
        propiedades
          esposo: hombre;
          string: nombre;
          ...
    fin Mujer
```

RESUMEN

Un método general para la resolución de un problema con computadora tiene las siguientes fases:

1. *Análisis del programa.*
2. *Diseño del algoritmo.*
3. *Codificación.*

El sistema más idóneo para resolver un problema es descomponerlo en módulos más sencillos y luego, mediante diseños descendentes y refinamiento sucesivo, llegar a módulos fácilmente codificables. Estos módulos se deben codificar con las estructuras de control de programación estructurada.

Los tipos abstractos de datos (**TAD**) describen un conjunto de objetos con la misma representación y comportamiento. Los tipos abstractos de datos presentan una separación clara entre la interfaz externa de un tipo de datos y su implementación interna. La implementación de un tipo abstracto de datos está oculta. Por consiguiente, se pueden utilizar implementaciones alternativas para el mismo tipo abstracto de dato sin cambiar su interfaz.

La especificación de un tipo abstracto de datos se puede hacer de manera *informal*, o bien, de forma más rigurosa, una especificación *formal*. En la especificación *informal* se describen literalmente los datos y la funcionalidad de las operaciones. La especificación formal describe los datos, la sintaxis y la semántica de las operaciones considerando ciertas operaciones como axiomas, que son los constructores de nuevos datos. Una buena especificación formal de un tipo abstracto de datos debe poder verificar la *bondad* de la implementación.

Las características más importantes de la programación orientada a objetos son: abstracción, encapsulamiento, herencia, polimorfismo, clases y objetos.

Una **clase** es un tipo de dato definido por el usuario que sirve para representar objetos del mundo real. Un objeto de una clase tiene dos componentes: un conjunto de atributos y un conjunto de operaciones. Un objeto es una instancia de una clase. Herencia es la relación entre clases que se produce cuando una nueva clase se crea utilizando las propiedades de una clase ya existente. Polimorfismo es la propiedad por la que un mensaje puede significar cosas diferentes dependiendo del objeto que lo recibe.

EJERCICIOS

1.1. Diseñar el diagrama de flujo de algoritmo que visualice y sume la serie de números 4, 8, 12, 16, ..., 400.

1.2. Diseñar un diagrama de flujo y un algoritmo para calcular la velocidad (en m/s) de los corredores de la carrera de 1.500 metros. La entrada consistirá en parejas de números (minutos, segundos) que dan el tiempo del corredor; por cada corredor, el algoritmo debe imprimir el tiempo en minutos y segundos así como la velocidad media. Ejemplo de entrada de datos: (3,53) (3,40) (3,46) (3,52) (4,0) (0,0); el último par de datos se utilizará como fin de entrada de datos.

1.3. Escribir un algoritmo que encuentre el salario semanal de un trabajador, dada la tarifa horaria y el número de horas trabajadas diariamente.