POO y Conceptos Básicos de las Estructuras de Datos

Luis Gerardo Montané Jiménez

Agosto 2017

Universidad Veracruzana

FACULTAD DE ESTADÍSTICA E INFORMÁTICA

Introducción

- El término Orientado a Objetos (OO) promueve que el software sea organizado como una colección de objetos que contienen datos y comportamientos
- Se busca hacer que el software sea más fácil de mantener, escribir y reutilizar
- Las características básicas de la programación OO (POO) son: Abstracción, Encapsulación, Polimorfismo y Herencia

Abstracción (1/2)

- La abstracción en la POO promueve el modelado centrado en aspectos esenciales de una entidad, ignorando sus propiedades no relevantes
- En la construcción de software significa centrarse en lo que es y lo que hace un objeto antes de decidir cómo debería ser implementada
- Para apoyar la construcción de sistemas bajo el paradigma OO han surgido modelos que ayudan la abstracción de un problema

Abstracción (2/2)

- El uso de modelos para la programación OO tiene como finalidad la abstracción de aquellos aspectos que sean importantes
- Un buen modelo Orientado a Objetos (OO) captura los aspectos cruciales del problema y omite los demás
- Un modelado de objetos captura la estructura estática del sistema
- El modelo de clases corresponde con el mudo real de manera más fiel

Clase

- Es una abstracción que permite definir un tipo de objeto, junto con propiedades (atributos) y operaciones (métodos)

- Es un elemento para la creación de objetos a partir de un modelo pre-definido

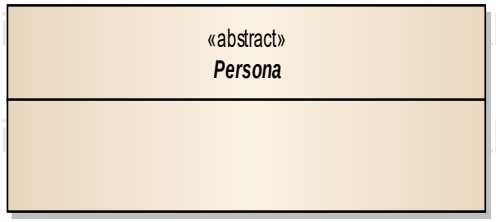
Objeto

- Entidad existente que tiene propiedades con datos del mismo objeto y operaciones específicas (métodos)

- Es el resultado de instanciación de una clase

Encapsulación

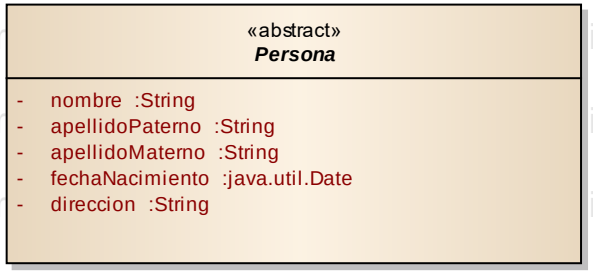
- La encapsulación es un mecanismo de programación por el que se establece una relación entre las operaciones y los datos que se manipulan
- En los lenguajes de programación OO es posible relacionar los datos y las operaciones en cajas negras independientes
- Dentro de un objeto los datos o las operaciones pueden ser privados o públicos
- El código privado únicamente es accedido desde adentro del objeto, mientras que lo público se puede acceder desde otro objeto
- La unidad básica de encapsulación es la clase



Atributos y Propiedades

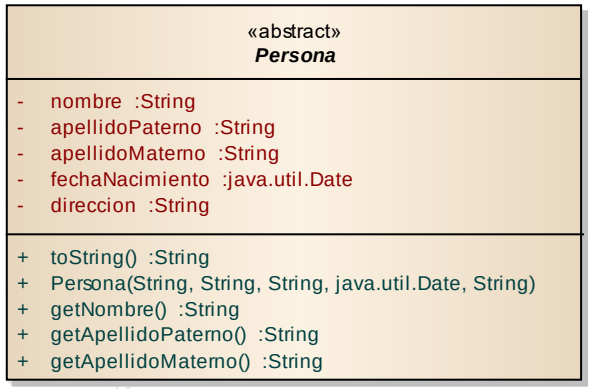
- Un atributo es un valor de un dato que está almacenado en los objetos de una clase, ejemplo de atributos son:
  - Nombre y fecha de nacimiento

- Cada atributo tiene un valor para cada instancia del objeto
- Las instancias distintas de una cierta clase pueden tener el mismo valor o valores distintos para un atributo dado

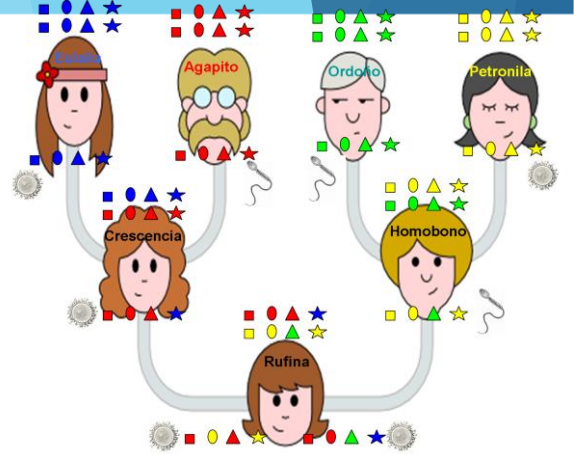


Operaciones

- Una operación es una función o transformación que se puede aplicar o que puede ser aplicada por los objetos de una clase
- Por ejemplo, prestar, devolver y reservar son operaciones de la clase Trabajador o Biblioteca
- Todos los objetos de una clase comparten las mismas operaciones

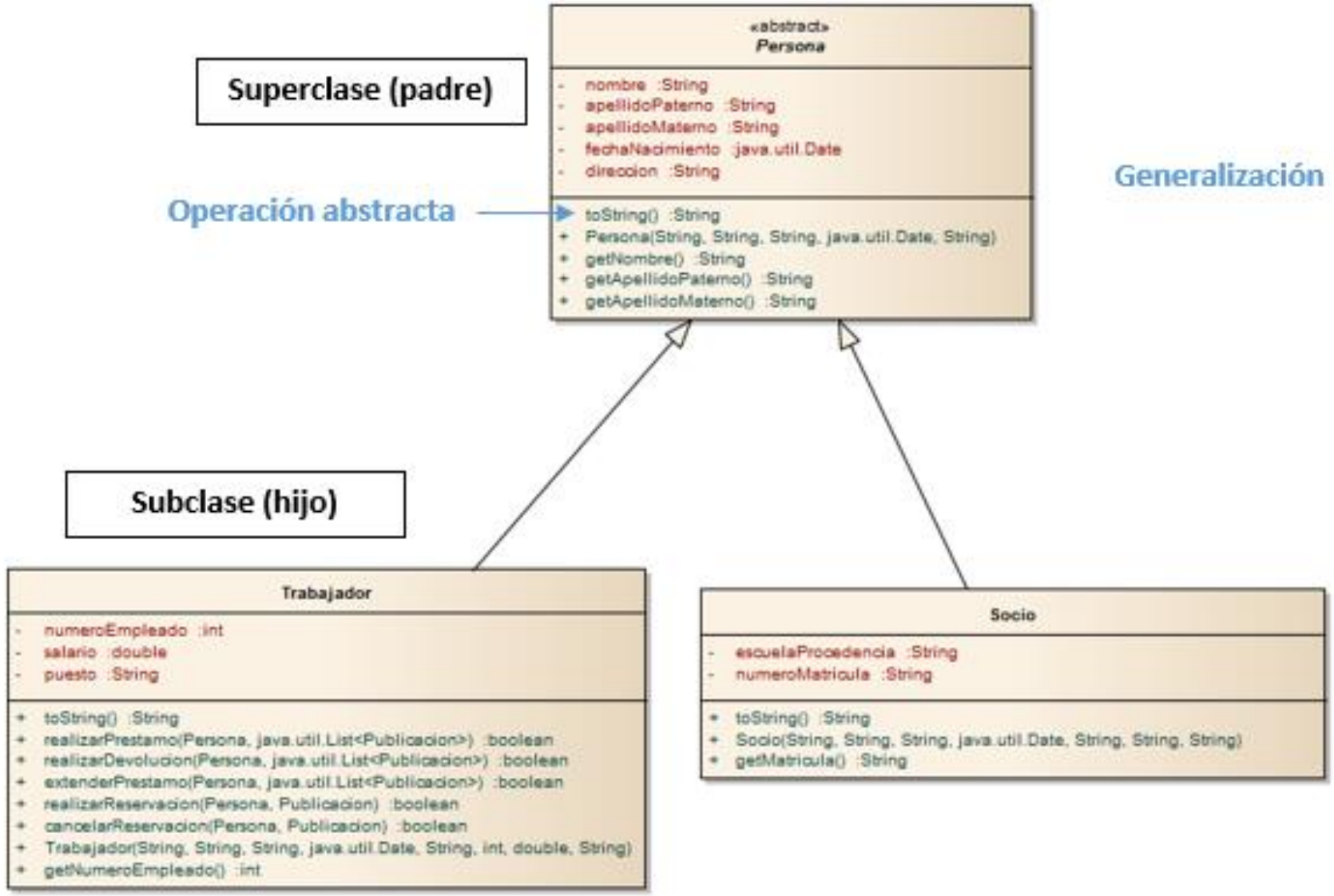


Herencia (1/3)

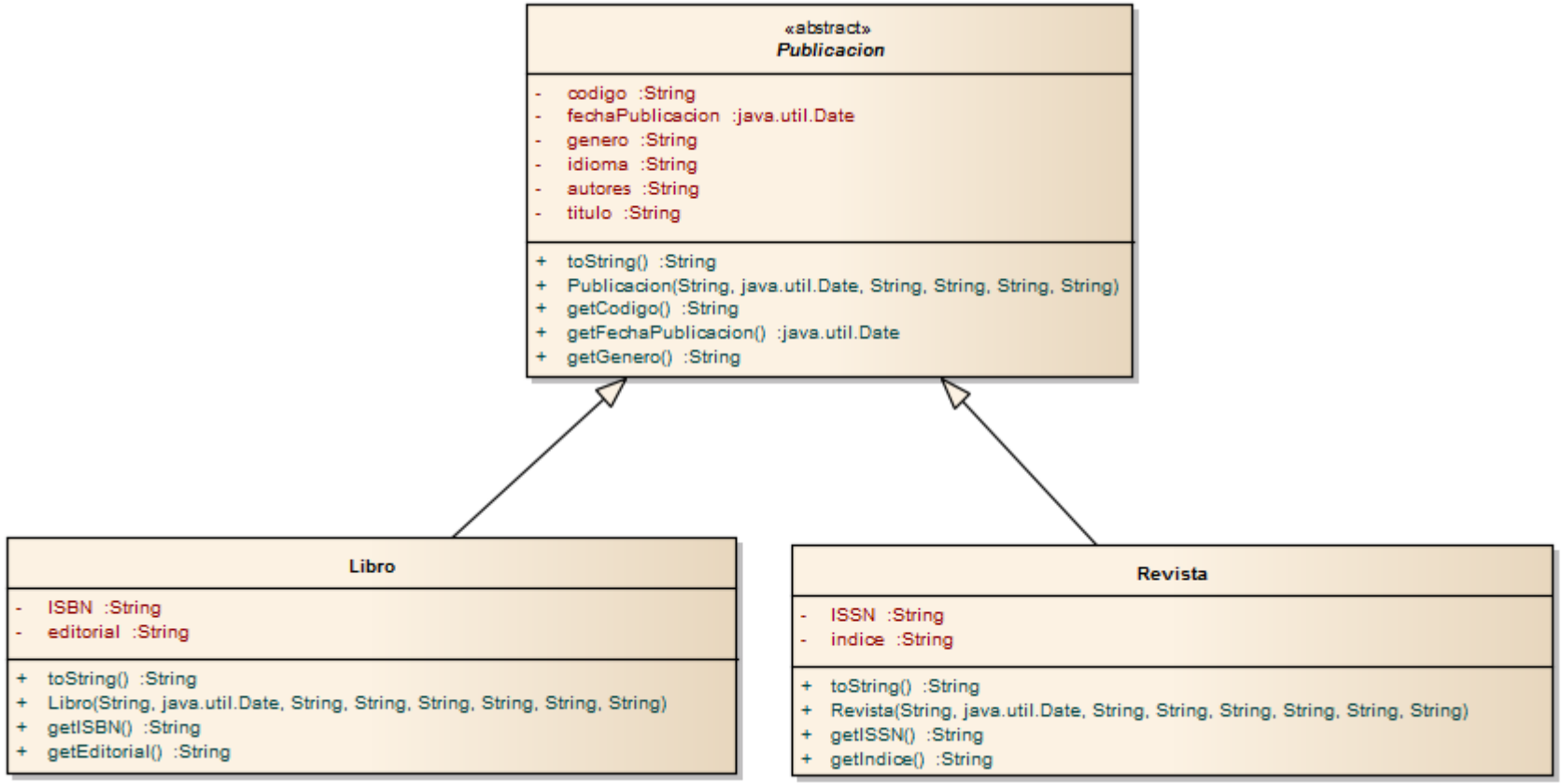


- La herencia es el proceso mediante el cual un objeto puede adquirir propiedades de otro
- La clase hereda las propiedades generales de su padre
- La herencia es el mecanismo que le permite a un objeto ser una instancia específica de una clase más general
- En este ámbito, se introducen los términos de *subclases* y *superclases*
- Las subclases contiene los atributos y métodos de la clase de la cual se deriva (superclase)
- La herencia es una potente abstracción para compartir similitudes entre clases
- Puede representarse visualmente de forma jerárquica, comenzando con la clase base llamada también superclase de la cual se derivan las clases secundarias

H


Superclase (padre)
«abstract»
Persona
- nombre :String
- apellidoPaterno :String
- apellidoMaterno :String
- fechaNacimiento :java.util.Date
- direccion :String
Operación abstracta
toString() :String
+ Persona(String, String, String, java.util.Date, String)
+ getNombre() :String
+ getApellidoPaterno() :String
+ getApellidoMaterno() :String
Generalización
Subclase (hijo)
Trabajador
- numeroEmpleado :int
- salario :double
- puesto :String
+ toString() :String
+ realizarPrestamo(Persona, java.util.List<Publicacion>) :boolean
+ realizarDevolucion(Persona, java.util.List<Publicacion>) :boolean
+ extenderPrestamo(Persona, java.util.List<Publicacion>) :boolean
+ realizarReservacion(Persona, Publicacion) :boolean
+ cancelarReservacion(Persona, Publicacion) :boolean
+ Trabajador(String, String, String, java.util.Date, String, int, double, String)
+ getNumeroEmpleado() :int
Socio
- escuelaProcedencia :String
- numeroMatricula :String
+ toString() :String
+ Socio(String, String, String, java.util.Date, String, String, String)
+ getMatricula() :String

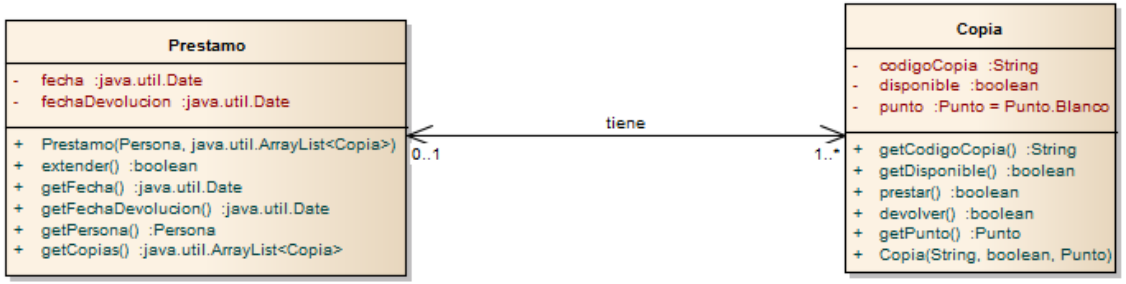
Herencia (3/3)


«abstract»
Publicacion
- codigo :String
- fechaPublicacion :java.util.Date
- genero :String
- idioma :String
- autores :String
- titulo :String
+ toString() :String
+ Publicacion(String, java.util.Date, String, String, String, String)
+ getCodigo() :String
+ getFechaPublicacion() :java.util.Date
+ getGenero() :String
Libro
- ISBN :String
- editorial :String
+ toString() :String
+ Libro(String, java.util.Date, String, String, String, String, String, String)
+ getISBN() :String
+ getEditorial() :String
Revista
- ISSN :String
- indice :String
+ toString() :String
+ Revista(String, java.util.Date, String, String, String, String, String, String)
+ getISSN() :String
+ getIndice() :String

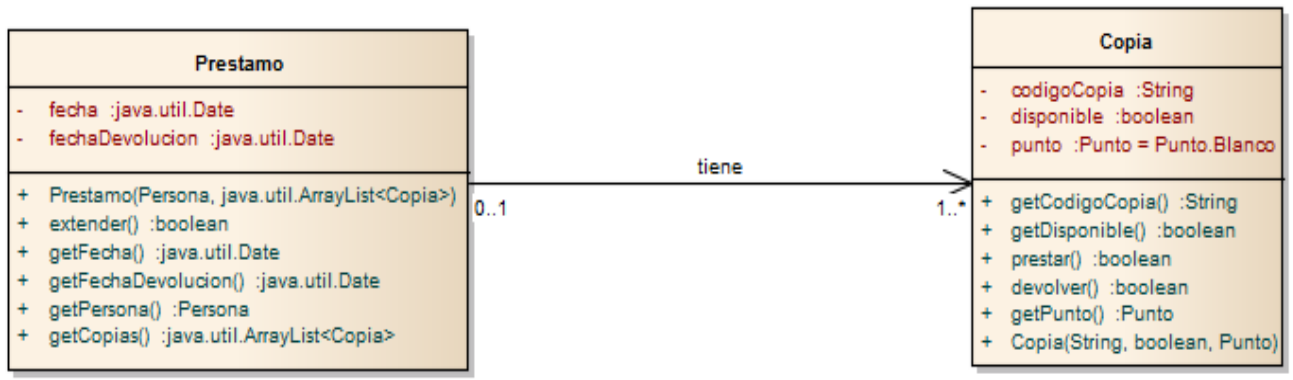
Asociaciones Bidireccionales y Unidireccionales

- Los enlaces muestran una relación entre dos (o más objetos)
- El modelado de un enlace oculta el hecho consistente en que el enlace no forma parte ninguno de los objetos por sí mismo, sino que depende de ambos a la vez

Asociaciones Bidireccionales

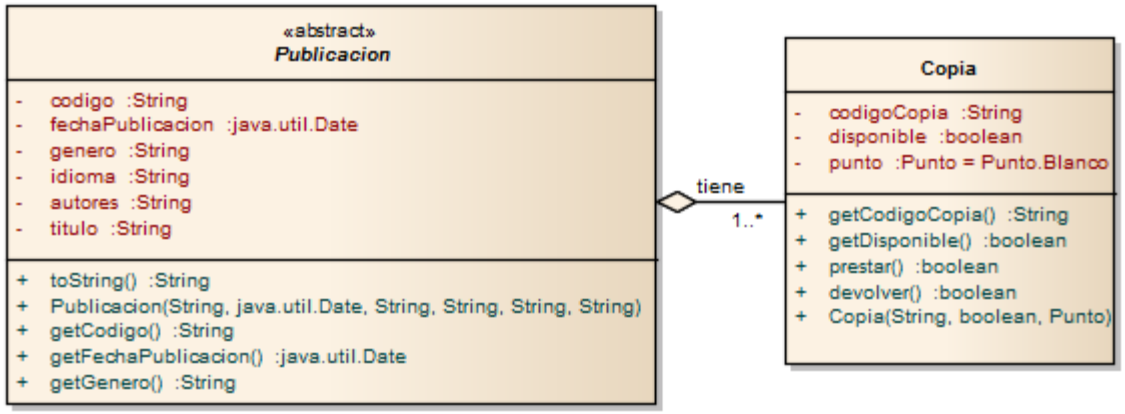


Asociación Unidireccional



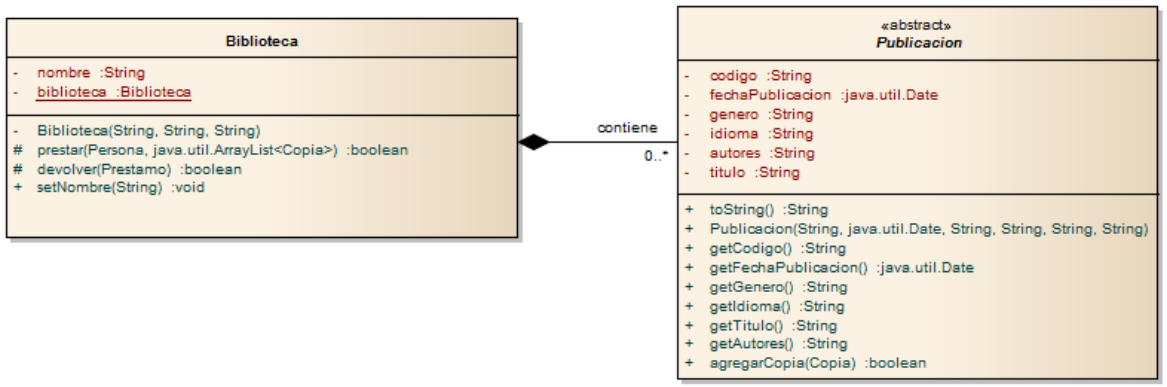
Agregación

- La agregación es la representación de una relación
- Para su representación se utiliza un diamante hueco en el extremo de la trayectoria unida a la clase agregada
- La vida del objeto agregado no depende del objeto compuesto



Composición

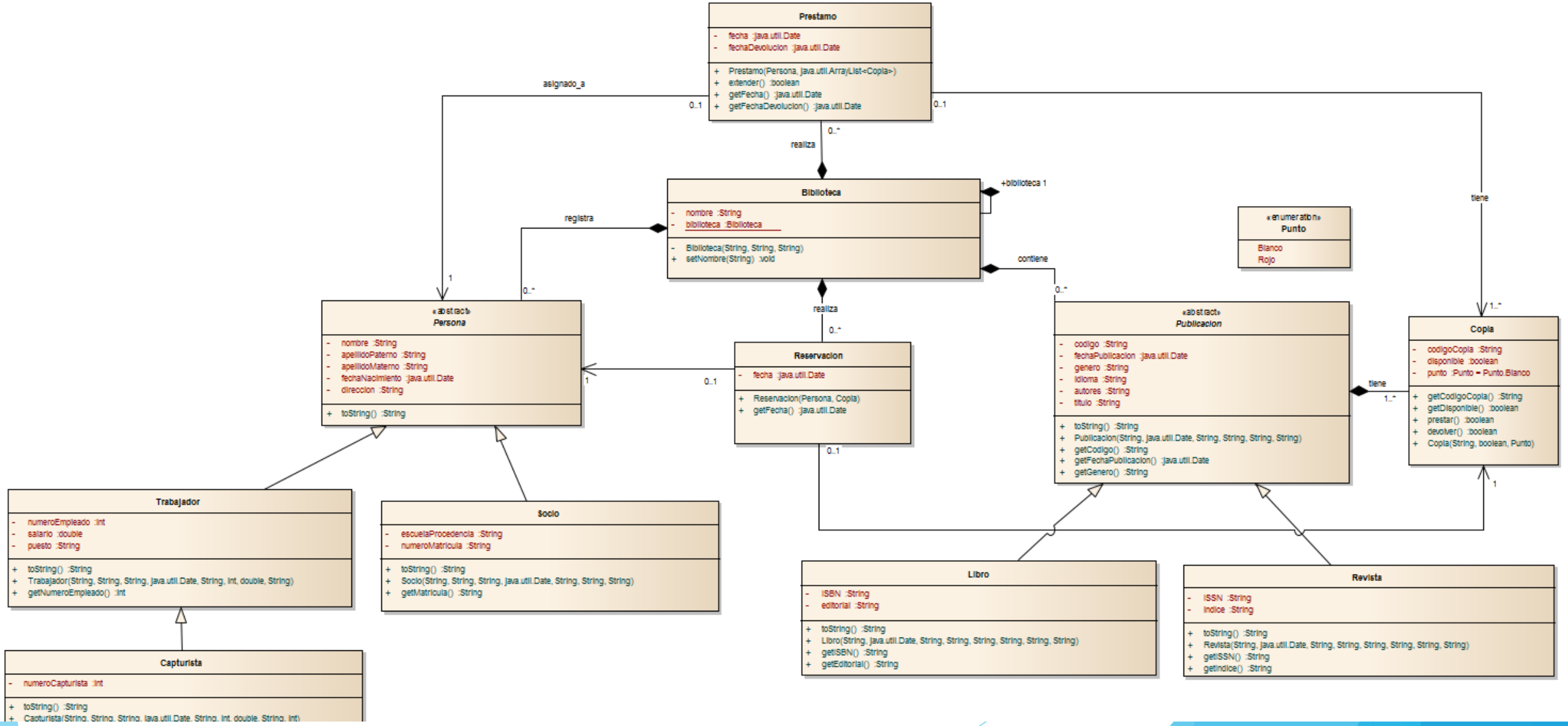
- Una composición es una forma más fuerte de asociación
- El objeto compuesto es el responsable único de gestionar sus partes
- En java, esta composición se puede implementar con una lista de *Publicaciones* agregada en la clase *Biblioteca*



Contenido

- Abstracción
- Encapsulación
- Herencia
- **Diagrama de Clases**

Diagrama de Clases



Práctica

Sistema de tipos

- El concepto de tipo es muy importante en C++
- Cada variable, argumento de función y valor devuelto por una función debe tener un tipo para compilarse
- Antes de evaluar cada una de las expresiones (incluidos los valores literales), el compilador da implícitamente un tipo a estas expresiones
- Algunos ejemplos de tipos son int, que almacena valores enteros, double, que almacena valores de punto flotante (también conocidos como tipos de datos escalares) o la clase std::basic_string de la biblioteca estándar, que almacena texto.

Sistema de tipos

- Puede crear su propio tipo definiendo un objeto **class** o **struct**
- El tipo especifica:
  - La cantidad de memoria que se asignará para la variable (o el resultado de la expresión)
  - Las clases de valores que se pueden almacenar en esa variable, cómo se interpretan estos valores (como patrones de bits) y las operaciones que se pueden realizar en ella.

Terminología

- **Variable**: nombre simbólico de una cantidad de datos.
  - Este nombre se puede utilizar para acceder a los datos a los que hace referencia en el ámbito del código en el que se define. En C++, el término “variable” se utiliza normalmente para hacer referencia a las instancias de tipos de datos.

- **Objeto**: por simplicidad y coherencia, el término “objeto” se usa para hacer referencia a cualquier instancia de una clase o estructura.

Especificar tipos de variable y función

- C++ es un lenguaje fuertemente tipado y que, además, contiene tipos estáticos. Cada objeto tiene un tipo y ese tipo nunca cambia (no debe confundirse con los objetos de datos estáticos)
- Al declarar una variable en el código, debe especificar explícitamente su tipo o utilizar la palabra clave auto para indicar al compilador que deduzca el tipo desde el inicializador
- Al declarar una función en el código, debe especificar el **tipo de cada argumento** y su **valor devuelto** (o void, si la función no devuelve ningún valor)
- La excepción se produce cuando se utilizan plantillas de función, que están permitidas en los argumentos de tipos arbitrarios
- Una vez que se declara por primera vez una variable, no se puede cambiar su tipo. Sin embargo, el valor de la variable o el valor devuelto por una función se puede copiar en otra variable de distinto tipo
  - Este tipo de operaciones se denominan conversiones de tipo. Estas conversiones a veces resultan necesarias, aunque también pueden producir errores o pérdidas de datos

Tipos de Datos

- Existen dos tipos
  - **Tipos simples llamados también primitivos**
  - **Tipos estructurados**

Tipos Simples (integrales)

- Son atómicos
- Ejemplos:
  - Número enteros, dobles (punto flotante), string de la biblioteca string.h para texto

- A diferencia de algunos lenguajes, C++ no tiene un tipo base universal del que se deriven todos los demás tipos. La implementación del lenguaje C++ contiene muchos tipos fundamentales, también conocidos como tipos integrados. Esto incluye los tipos numéricos, como **int, double, long y bool**

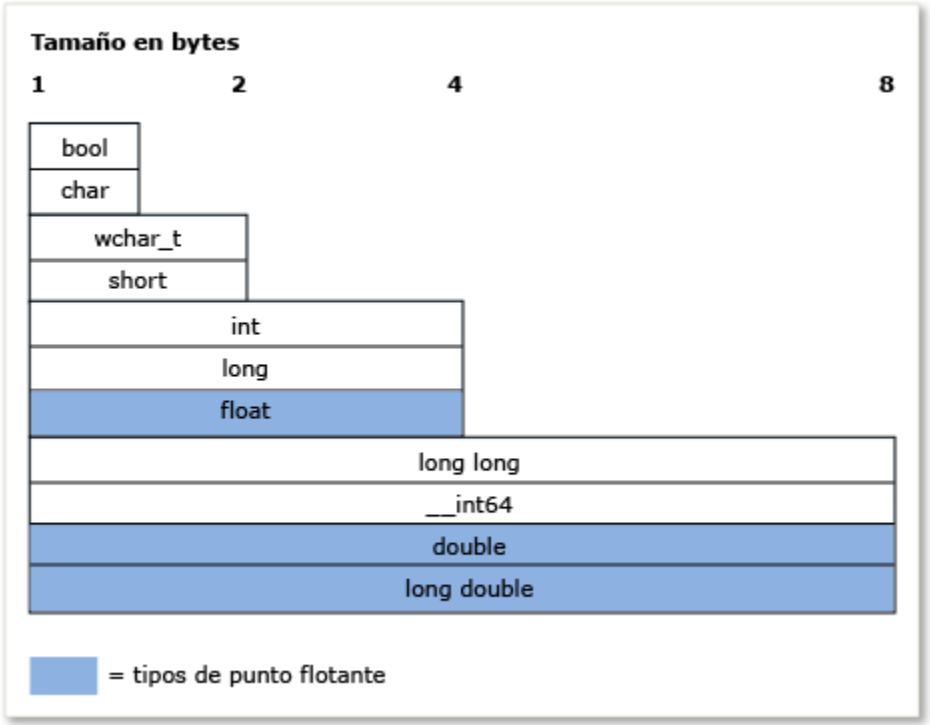
Tipos Simples (integrales)

La mayoría de los tipos fundamentales (excepto bool, double y tipos relacionados) tienen versiones **sin signo**, que modifican el intervalo de valores que la variable puede almacenar.

Por ejemplo, un valor int, que almacena un entero de **32 bits con signo**, puede representar un valor comprendido entre -2.147.483.648 y 2.147.483.647.

Un valor **unsigned int**, que también se almacena como **32 bits**, puede almacenar un valor comprendido entre 0 y 4.294.967.295. El número total de valores posibles en cada caso es el mismo; solo cambia el intervalo.

Tamaño relativo de los tipos



Tamaño relativo de los tipos

Tipo	Tamaño	Comentario
int	4 bytes	Opción predeterminada para los valores enteros.
double	8 bytes	Opción predeterminada para los valores de punto flotante.
bool	1 byte	Representa valores que pueden ser true o false.
char	1 byte	Se utiliza en los caracteres ASCII de cadenas de estilo C antiguas u objetos std::string que nunca tendrán que convertirse a UNICODE.
wchar_t	2 bytes	Representa valores de caracteres "anchos" que se pueden codificar en formato UNICODE (UTF-16 en Windows; puede diferir en otros sistemas operativos). Es el tipo de carácter que se utiliza en las cadenas de tipo `std::wstring`.
unsigned char	1 byte	C++ no tiene un tipo `byte` integrado. Utilice un carácter sin signo para representar un valor byte.
unsigned int	4 bytes	Opción predeterminada para los marcadores de bits.
long long	8 bytes	Representa valores enteros muy grandes.

Tamaño de los tipos enteros en rangos

- int (4 bytes): -2147483648 a 2147483647
- short int (2 bytes): -32768 a 32767
- long (long int): 8 bytes: -9223372036854775808 a 9223372036854775808
- unsigned int (4 bytes): 0 a 4294967295

Números enteros

- El más utilizado es el **int**
- Un entero largo es de tipo **long**

- Los siguientes modificadores permiten una definición más específica:
  - signed: indica que el valor máximo que se puede representar tiene por punto medio el 0 (por lo tanto se pueden representar valores negativos)
  - int es por defecto signed
  - unsigned: utilizado para poder representar números positivos
  - short: tipo con menos bytes
  - long: tipo que reserva más byte