Fundamentos de la Programación Orientada a Objetos

Luis Gerardo Montané Jiménez

Febrero 2018

Universidad Veracruzana

FACULTAD DE ESTADÍSTICA E INFORMÁTICA

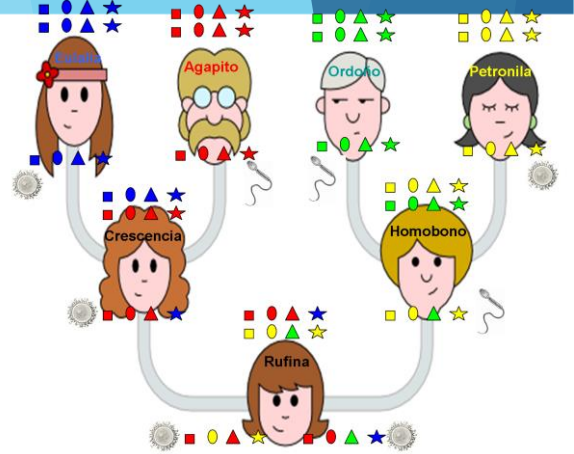
Objetivo

- Abordar el concepto de herencia de la Programación Orientada a Objetos

Contenido

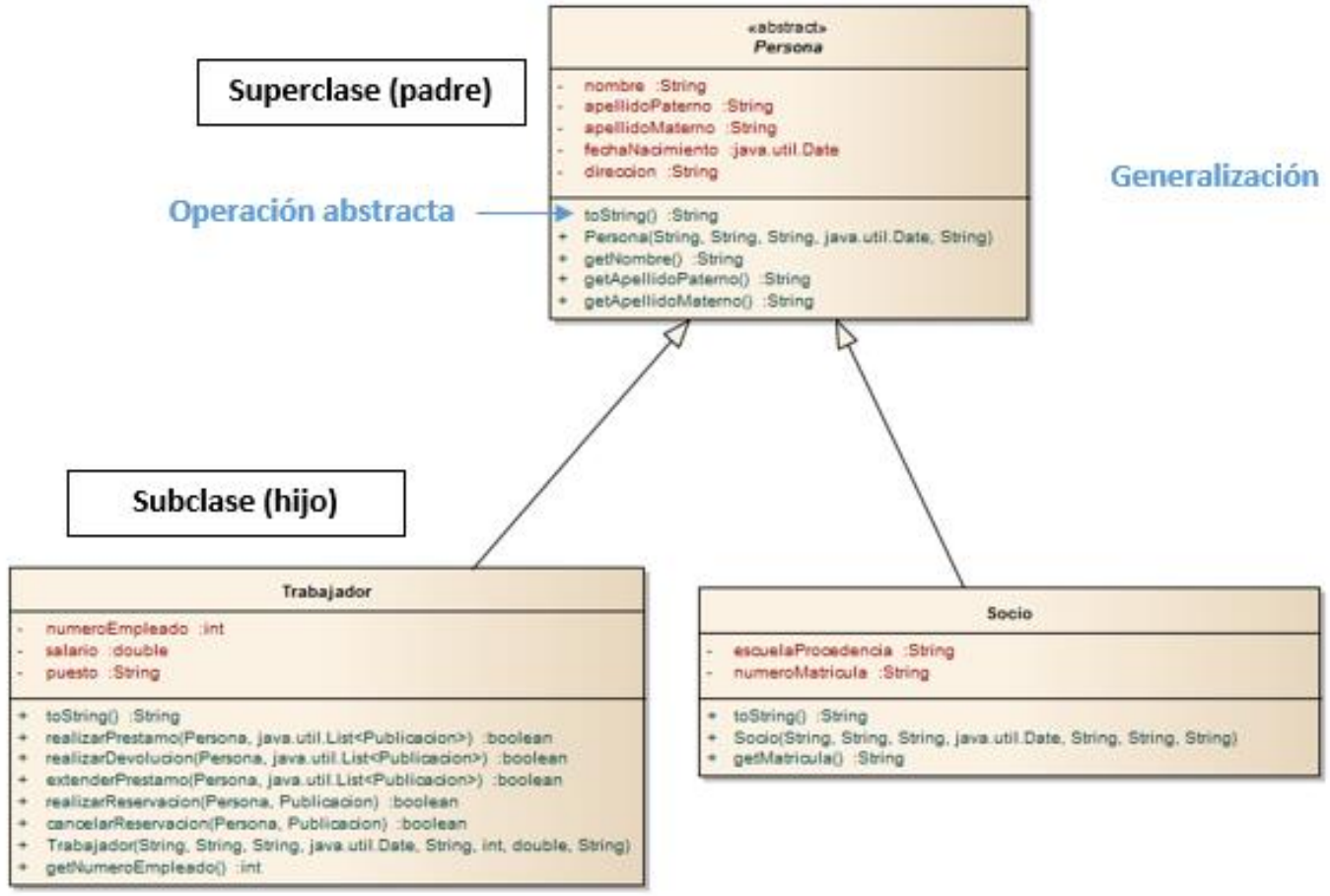
- Abstracción
- Encapsulación
- **Herencia**

Herencia (1/3)

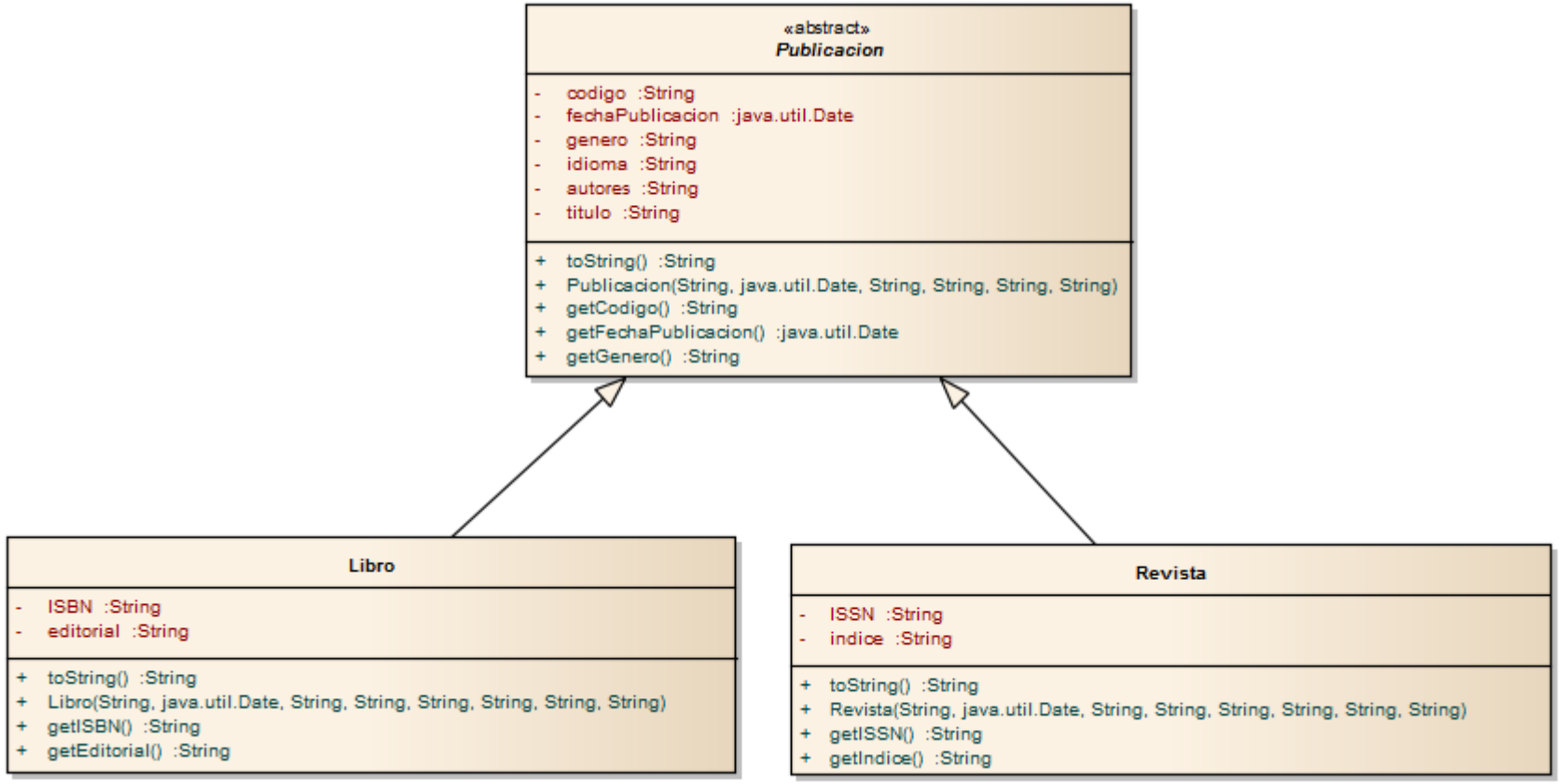


- La herencia es el proceso mediante el cual un objeto puede adquirir propiedades de otro
- La clase hereda las propiedades generales de su padre
- La herencia es el mecanismo que le permite a un objeto ser una instancia específica de una clase más general
- En este ámbito, se introducen los términos 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
- Los constructores no se heredan

Herencia (2/3)

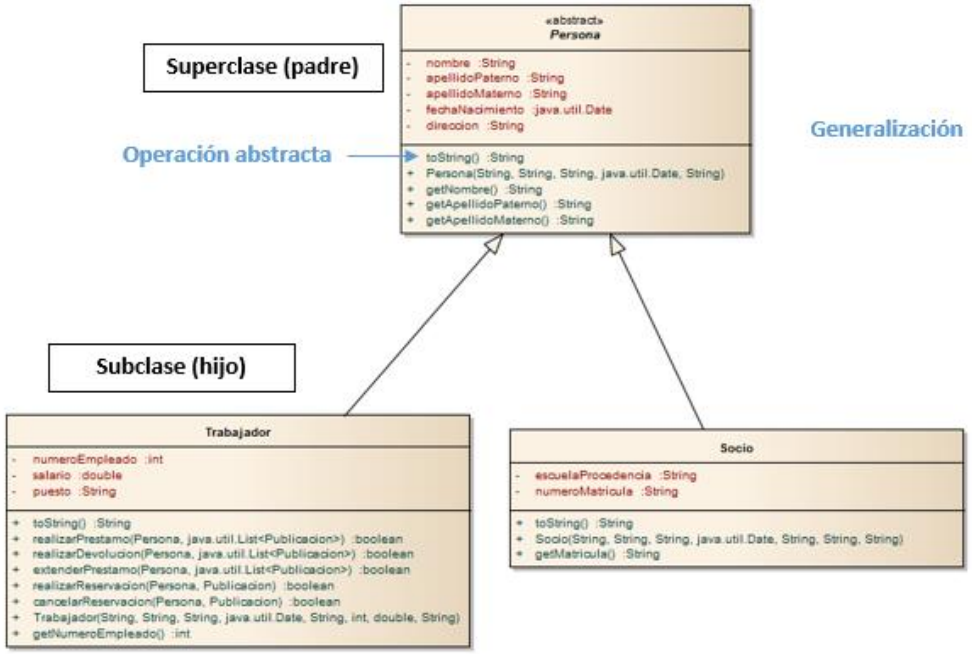

Superclase (padre)
Operación abstracta
Generalización
Subclase (hijo)
«abstract»
Persona
- nombre :String
- apellidoPaterno :String
- apellidoMaterno :String
- fechaNacimiento :java.util.Date
- direccion :String
+ toString() :String
+ Persona(String, String, String, java.util.Date, String)
+ getNombre() :String
+ getApellidoPaterno() :String
+ getApellidoMaterno() :String
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

Alcance de la Herencia

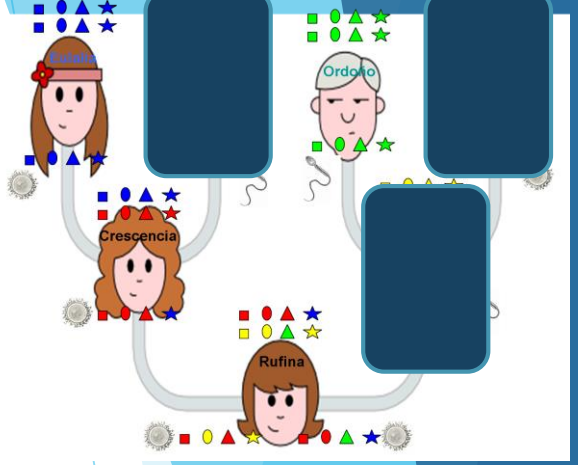
- La herencia es transitiva: una clase puede heredar características de superclases que se encuentran muchos niveles más arriba en la jerarquía de herencia
- La herencia es clasificada con la dimensión *variedad*:
  - A **ES-UN** B, y donde la clase A se relaciona con B por herencia
  - Ejemplos:
    - Un Trabajador es una Persona
    - Un Socio es una Persona
    - Un Perro es un Animal
    - Un Auto es un Vehículo
    - Un Motor es un Vehículo ¿Incorrecto?



Implementación en Java

- Sobre-escritura de métodos
  - Consiste en cambiar el comportamiento de un método que tiene la misma firma en una clase derivada
- Si el método de la clase base se pretende que no sea sobre-escrito entonces el método debe ser declarado utilizando la palabra reservada **final**
- Ejemplo:
  - Si la clase Persona es la clase base (superclase, padre) y se busca tener una clase derivada entonces la herencia se realiza de la siguiente forma:
    - Superclase: class Persona{….}
    - Clase derivada (subclase): class Trabajador **extends** Persona{….}
    - Clase derivada (subclase): class Socio **extends** Persona{….}
    - Para acceder a elementos de la clase base se utiliza la palabra reservada **super**.

Herencia Múltiple



- En caso de que una clase tenga más de un padre, hereda de ambos
- Estas propiedades (atributos, operaciones) son la unión de los padres
- En lenguajes de programación como Java no es posible implementar la herencia múltiple

Sobre-carga de métodos (1/3)

- La firma de un método es la combinación del tipo de dato que regresa, su nombre y lista de parámetros

- La sobre-carga de métodos es la creación de varios métodos con el mismo nombre pero con diferentes firmas (nombre del método, cantidad, tipo y orden de parámetros)

```
int calcular(int x, int y, int z){
    ...
}
int calcular(double x, double y, double z){
    ...
}
```

- El tipo de valor de retorno no forma parte de la firma del método (no se utiliza para distinguir entre métodos)

```
int calcular(int x, int y, int z){…}
double calcular(int x, int y, int z){…}
```

Sobre-carga de métodos (2/3)

- Los métodos sobrecargados poseen el mismo nombre sin importar el numero de métodos que existan
- Se puede usar cualquier tipo de método (String, int, float, double, etc….)
- En caso de que el método sea diferente de **void** se debe de retornar un valor dependiendo del tipo de método declarado
- Los parámetros o argumentos que posean los métodos sobrecargados pueden ser de diferentes tipos y diferente cantidad de estos

Sobre-escritura de métodos (3/3)

- La subclase o clase derivada hereda todos los métodos de su superclase que son accesibles a dicha subclase a menos que sobreescriba los métodos

- La subclase sobreescribe un método de la superclase cuando define un método con la misma firma o características (nombre, número y tipo de parámetros) que el método de la superclase

- La sobreescritura de métodos en las subclases es utilizada tradicionalmente para agregar o modificar la funcionalidad del método heredado de la clase padre

Sobre-carga de constructores

- La sobrecarga de constructores es cuando en una clase existen constructores múltiples

```
Constructor()
{
   nombre = null;
   edad = 0;
   direccion = null;
}

Constructor(String nombre, int edad, String direccion)
{
   this.nombre = nombre;
   this.edad = edad;
   this.direccion = direccion;
}
```