Universitat Pompeu Fabra Barcelona

Seminario 4

Universitat Pompeu Fabra Barcelona

El objetivo del seminario es:

Aprender a diseñar una aplicación que utilice el concepto de interfaces. En concreto, se utilizarán las interfaces para ordenar las tablas de los equipos de una liga y clasificar a los máximos goleadores de una competición.

La solución a estos ejercicios se implementará en Java durante la sesión 4 del laboratorio.

Nota del seminario

- Relaciones (50%): tipo correcto de relación, notación correcta
- Cardinalidad (10%)
- Atributos (15%): incluyen los atributos correspondientes a las *clases asociación*
- Métodos (15%): incluyen los constructores
- Tipos (5%): de atributos, argumentos de métodos, tipos de devolución, etc.
- Visibilidad (5%): public, private

Clase Abstracta

Una clase Abstracta es similar a una clase normal, la estructura es prácticamente igual, ya que poseen nombre, atributos y métodos pero para que una clase sea abstracta la condición es que al menos uno de sus métodos sea abstracto (se le agrega la palabra reservada **abstract**).

- Una clase Abstracta No puede ser instanciada (no se pueden crear objetos directamente - new), solo puede ser heredada.
- Si al menos un método de la clase es **abstract**, esto obliga a que la clase completa sea definida **abstract**, sin embargo la clase puede tener el resto de métodos no abstractos.
- Los métodos **abstract** no llevan cuerpo (no llevan los caracteres {}).
- La primer subclase concreta que herede de una clase **abstract** debe implementar todos los métodos de la **superclase**.

Interfaces

Una Interfaz es una Clase completamente Abstracta, como regla, sabemos que las **clases abstractas** poseen como **mínimo un método abstracto**, pero hablando de una **interfaz**, **todos sus métodos son abstractos**

Cuando se crea un **Interfaz**, lo que se hace es **definir** lo que la clase que la implemente podrá hacer, pero **no se indicará** la forma en que lo hará.

Las interfaces simulan la **herencia múltiple** ya que una clase puede implementar cualquier número de interfaces, además las interfaces pueden heredar uno o más números de interfaces mediante la palabra **extends**.

En java se usa la palabra reservada **implements** para indicar que se implementa una interfaz.

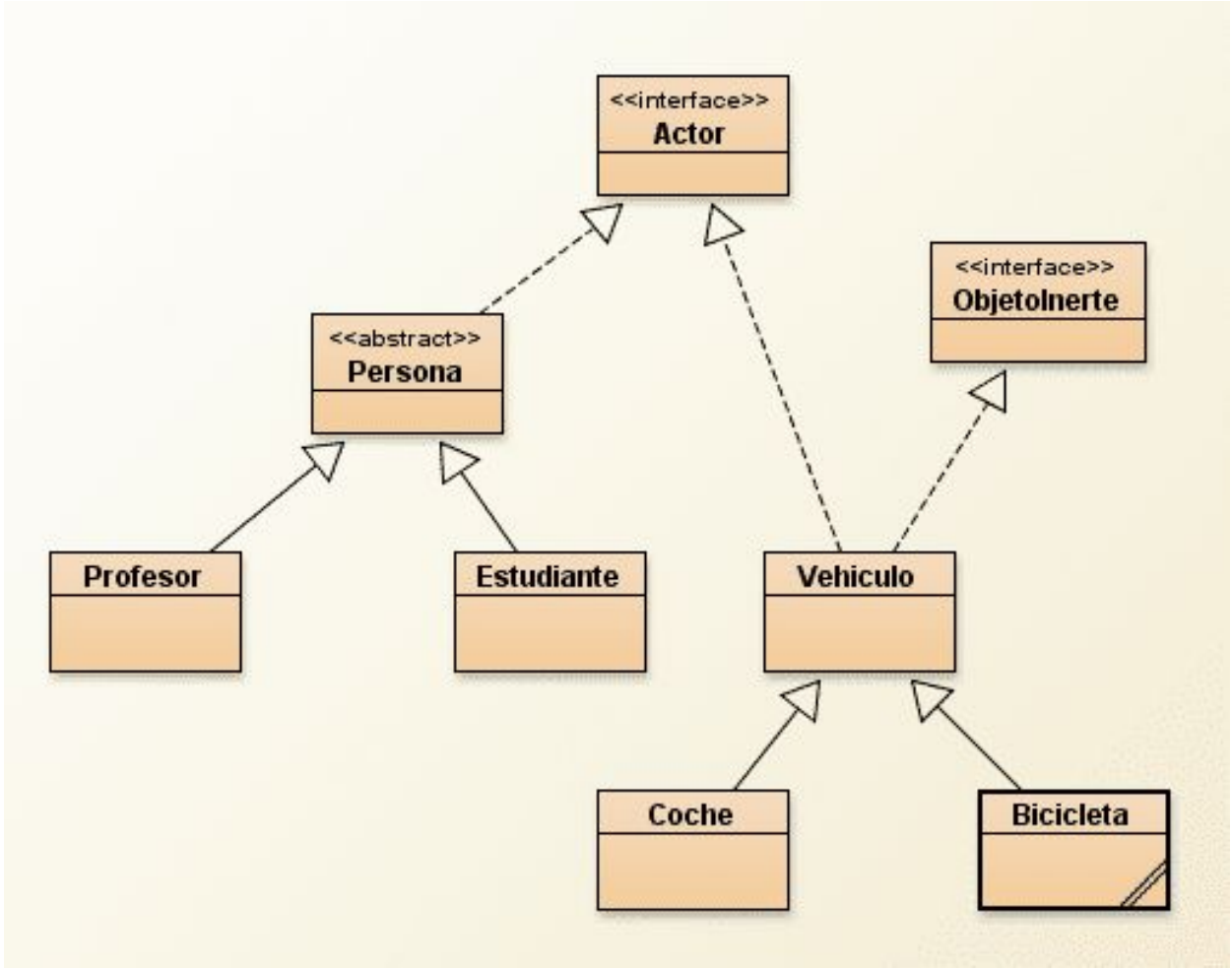
Clases abstractas o Interfaces

Existen varias **diferencias** entre una clase abstracta y una interfaz

- Una **clase abstracta** puede **heredar o extender cualquier clase** (independientemente de que esta sea abstracta o no), mientras que una **interfaz** solamente puede **extender o implementar otras interfaces.**
- Una **clase abstracta puede heredar de una sola clase** (abstracta o no) mientras que una **interfaz puede extender varias interfaces de una misma vez.**
- Una c**lase abstracta** puede tener **métodos** que sean **abstractos o que no** lo sean, mientras que las **interfaces** sólo y **exclusivamente** pueden definir **métodos abstractos.**

Clases abstractas o Interfaces

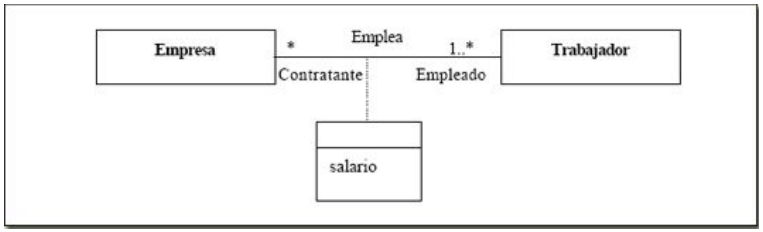
- En las **clases abstractas** la **palabra abstract** es obligatoria para definir un método abstracto (así como la clase).
- Cuando se define una **interfaz**, esta **palabra es opcional** ya que se infiere en el concepto de interfaz.
- En una **clase abstracta**, los **métodos** abstractos pueden ser **public o protected.** En una **interfaz solamente** puede haber métodos **públicos**.
- En una **clase abstracta** pueden existir **variables static, final o static final** con cualquier modificador de acceso (public, private, protected o default). En una **interfaz sólo** puedes tener **constantes** (public static final).


<<interface>>
Actor
<<interface>>
ObjetoInerte
<<abstract>>
Persona
Profesor
Estudiante
Vehiculo
Coche
Bicicleta

Universitat Pompeu Fabra Barcelona

Graphical representation

Asociación de clases



- The cardinality between A and B is defined in both directions
  - One-to-one: $\text{Card}_{B,A} = 1$, $\text{Card}_{A,B} = 1$
  - One-to-many: $\text{Card}_{B,A} = 1$, $\text{Card}_{A,B} = n$
  - Many-to-many: $\text{Card}_{B,A} = m$, $\text{Card}_{A,B} = n$
  - Zero-to-one (optional): $\text{Card}_{B,A} = 0/1$, $\text{Card}_{A,B} = 1$

- There exist five fundamental types of relations:
  1. Generalization/specialization or inheritance (is a)
  2. Composition/aggregation (is part of)
  3. Use/dependency (uses)
  4. Association (general relation)
  5. Template/generics

Dependency

Aggregation

Inheritance

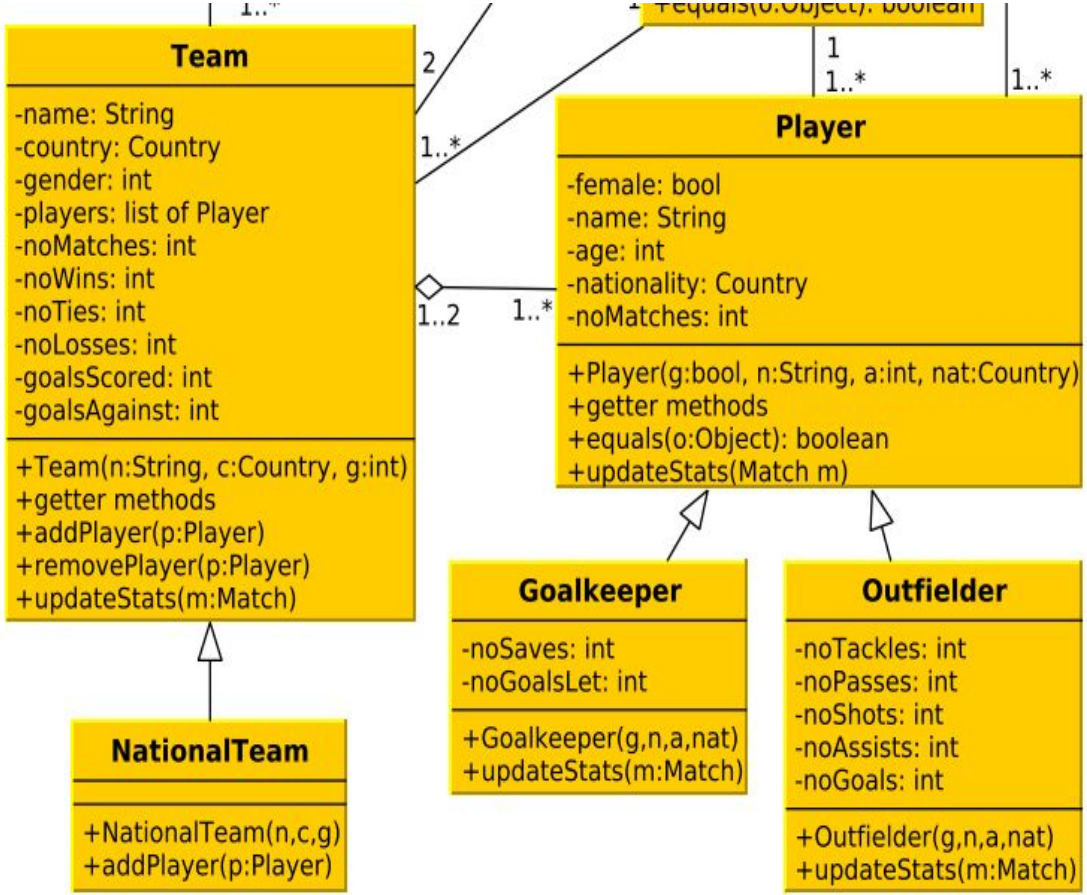
Composition

Association

Directed Association

Interface Type Implementation

Las clases



- En este seminario no es necesario repetir todas las clases del seminario anterior (la mayoría de ellos permanecerán sin cambios).
- Basta con diseñar un subdiagrama que incluya las clases relacionadas con equipos y estadísticas de equipos, así como jugadores y estadísticas de jugadores.

Ordenar las tablas de League

Hasta ahora, las estadísticas del equipo han sido parte de la clase Team. Sin embargo, esto es poco práctico, ya que un mismo equipo puede participar en múltiples competiciones durante un tiempo, por lo que, se hace necesario almacenar las estadísticas del equipo para cada competición.

Por ejemplo, si un equipo participa en una liga durante la temporada 2022-23 y durante la temporada 2023-24, se tendrá que realizar un seguimiento del número de victorias, empates y derrotas de cada liga por separado.

Ordenar las tablas de League

Dado que las estadísticas del equipo deben estar separadas del propio equipo, tiene sentido diseñar una nueva clase especialmente para este propósito y relacionar esta clase con el equipo existente.

Piense en cómo podemos asegurarnos de que las estadísticas del equipo sean asociadas correctamente a cada liga y cada equipo.

Ordenar las tablas de League

Cada equipo necesitará un método que actualice las estadísticas de una competición y un partido determinados.

Las estadísticas del equipo pueden necesitar acceso a la información sobre el equipo (como el nombre).

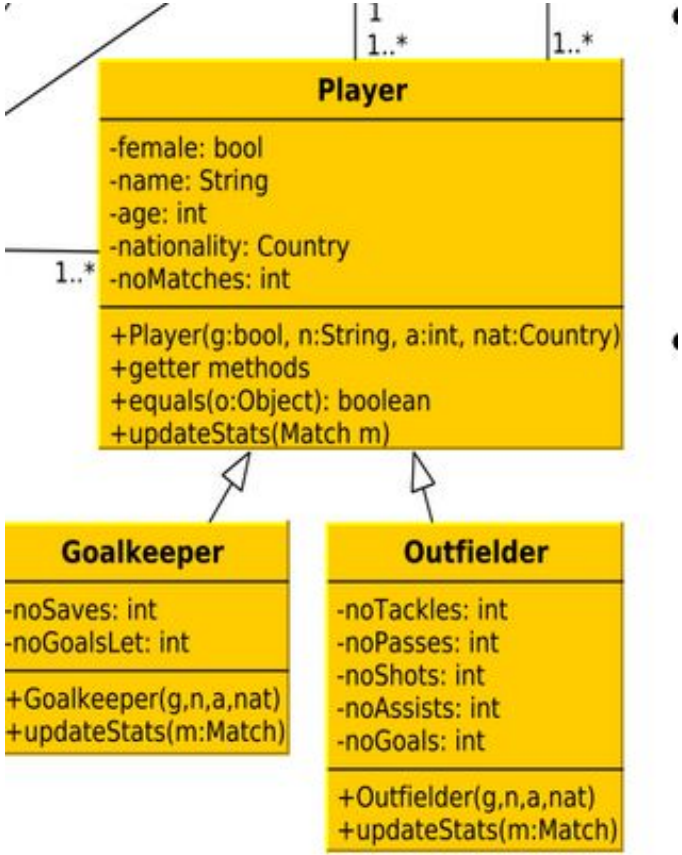
Las clases

Además, se puede utilizar la **nueva clase** para comparar las estadísticas de dos equipos.

Java contiene un método Collections:sort para ordenar colecciones de elementos. Para ordenar colecciones cuyos elementos son instancias de una clase T, es necesario que **T implemente la interfaz Comparable**, que contiene un único método abstracto **int compareTo(Object o)**. Se explicarán los detalles de este método en la sesión de laboratorio.

Clasificación de goleadores

Hasta ahora las estadísticas de los jugadores han formado parte del Portero y del jugador de campo, según el tipo de jugador. Sin embargo, un jugador puede también participar en múltiples competiciones a lo largo del tiempo, por lo que se tendrá que almacenar las estadísticas de los jugadores de cada competición.



Clasificación de goleadores

Al igual que las estadísticas del equipo, debería ser posible relacionar las estadísticas de los jugadores con un jugador determinado y una competición determinada.

Tener en cuenta que las estadísticas del jugador pueden necesitar acceso a información sobre el jugador (como el nombre).

Clasificación de goleadores

Además, dado que existen diferentes tipos de jugadores, es necesario almacenar diferentes tipos de estadísticas de jugadores (por ejemplo, estadísticas de portero y estadísticas de jugador de campo).

Cada tipo de jugador debe tener estadísticas apropiadas de acuerdo al tipo de jugador.

Cada jugador necesitará un método que actualice las estadísticas de una competición y de un partido determinado.

Clasificación de goleadores

Para clasificar a los goleadores, podemos implementar nuevamente la interfaz Comparable y sobreescribir el método abstracto int compareTo(Object o).

Los detalles de cómo implementar el mecanismo de clasificación se explicará en la sesión de laboratorio.