



# GUÍA PRÁCTICA PARA EL DISEÑO DE ONTOLOGÍAS CON PROTÉGÉ

---

Dr. Mikel Emaldi Manrique (m.emaldi@deusto.es)

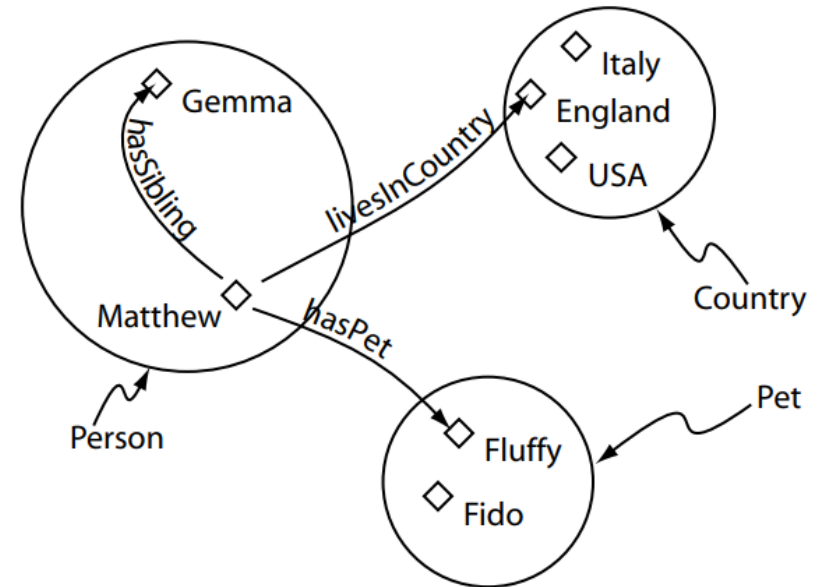
# CRÉDITOS

Presentación basada en el tutorial [“A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3”](#) por Matthew Horridge, Copyright The University of Manchester.

Todas las figuras están extraídas de este manual y son creación original de sus autores.

# COMPONENTES DE LAS ONTOLOGÍAS




- **Individuals (instancias):** representan objetos del dominio en el cual estamos interesados.
  - Podemos referirnos a ellos como “instancias de clases”.
- **Propiedades:** relaciones binarias entre instancias.
- **Clases:** conjuntos de instancias.
  - De describen de manera formal, especificando las condiciones que una instancia tiene que satisfacer para pertenecer a esa clase.
  - Se pueden organizar en jerarquías superclase-subclase.
  - Las subclases especializan sus superclases.

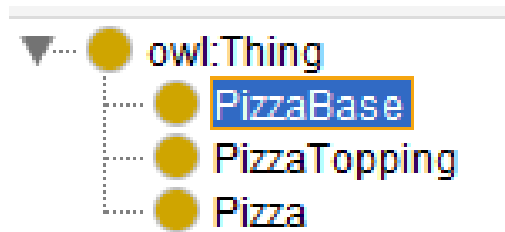


# CREACIÓN DE UNA NUEVA ONTOLOGÍA

- Descargar e instalar Protégé desde <https://protege.stanford.edu/products.php#desktop-protege>
- Abrir Protégé.
- Asignar una URL única para nuestra ontología, p. ej. *http://www.pizza.com/ontologies/pizza.owl*
- Guardar la ontología en un fichero en nuestro disco duro, p. ej. *pizza.owl*
- Buenas prácticas:
  - Indicar algunas anotaciones sobre la ontología que estamos creando.
  - Son importantes *rdfs:label*, *rdfs:comment* y *dc:license*

# CREACIÓN DE (NAMED) CLASES

- La clase *owl:Thing* representa a todas las instancias del dominio.
  - Todas las clases son subclase de *owl:Thing*.
- Ejercicio: crear las clases *Pizza*, *PizzaTopping* y *PizzaBase*
  - Acceder a la pestaña Entities → Clases.
  - Utilizar los botones    para crear subclases, clases hermanas o eliminar clases.



# CLASES DISJUNTAS

- En OWL las clases se solapan: no podemos decir que una instancia no pertenezca a una clase por el hecho de no haberlo dicho explícitamente.
- Podemos decir que las clases creadas son **disjuntas (disjoint)**, ya que una instancia no puede pertenecer a más de una de estas clases al mismo tiempo.
- Ejercicio: hacer que las clases creadas sean disjuntas.
  - Seleccionar una de las clases y en el panel que describe la clase, pinchar en el símbolo + en *Disjoint With*.
  - En la pestaña *Class Hierarchy*, seleccionar las clases que queremos que sean disjuntas.
    - Tip: con la tecla CTRL podemos seleccionar más de una.

## Description: PizzaBase

Equivalent To +

SubClass Of +


General class axioms +

SubClass Of (Anonymous Ancestor)

Instances +

Target for Key +

Disjoint With +

 **Pizza, PizzaTopping**

Disjoint Union Of +

# JERARQUÍA DE CLASES (I)

- Maneras más rápidas de crear jerarquías de clases:
  1. Botón derecho sobre la clase principal, “Add subclasses”
    - Podemos añadir una subclase por línea.
    - Al terminar nos pregunta si estas nuevas clases deberían ser disjuntas (suele ser lo más lógico).
  2. Seleccionar la clase principal, menú “Tools → Create class hierarchy”.
    - Podemos crear una jerarquía de clases a diferentes niveles, utilizando el tabulador.
    - Al terminar nos pregunta si estas nuevas clases deberían ser disjuntas (suele ser lo más lógico).
- Ejercicio: crear las subclases de *PizzaBase* *ThinAndCrispyBase* y *DeepPanBase*.

# JERARQUÍA DE CLASES (II)

- Ejercicio: crear la siguiente jerarquía de *PizzaToppings*:

Create Class Hierarchy

Enter hierarchy

Please enter the hierarchy that you want to create. You should use tabs to indent names!

Prefix

Suffix

Cheese

- Mozzarella
- Parmezan

Meat

- Ham
- Pepperoni
- Salami
- SpicyBeef

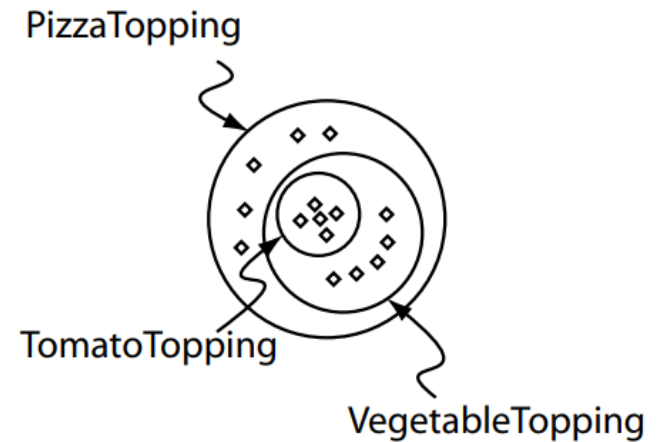
Seafood

- Anchovy
- Prawn
- Tuna

Vegetable

- Caper
- Mushroom
- Olive
- Onion
- Pepper
  - RedPepper
  - GreenPepper
  - JalapenoPepper
- Tomato

Go Back Continue Cancel





# CREACIÓN DE PROPIEDADES (OBJECT PROPERTIES)

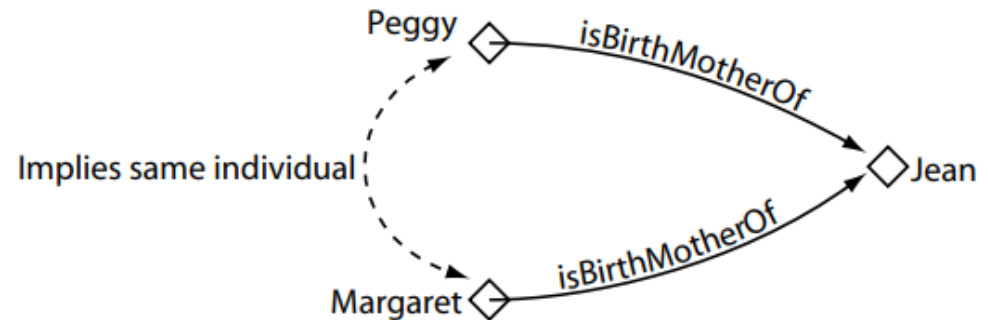
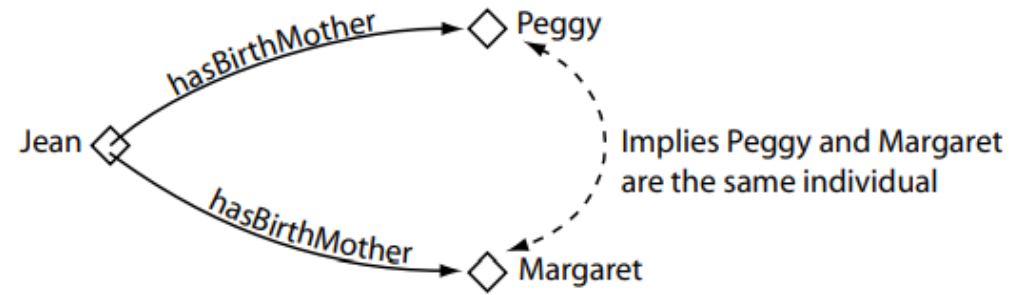
- Las **propiedades** OWL representan relaciones.
  - **Object properties**: relaciones entre dos instancias.
  - **Datatype properties**: relaciones entre una instancia y un XSD datatype.
  - **Annotation properties**: metadatos sobre las clases y las propiedades.
- Se encuentran en la pestaña “Object properties”
- Ejercicio: crear la propiedad *hasIngredient*.
- También existe jerarquía entre las *object properties*.
- Ejercicio: crear las propiedades *hasBase* y *hasTopping* como subclases de *hasIngredient*.

# PROPIEDADES INVERSAS (OBJECT PROPERTIES)

- Toda object property **debería** tener su **propiedad inversa (inverse property)**.
- Nos permiten que la ontología sea más formal y poder razonar sobre ella.
- Ejercicio: crear la propiedad *isIngredientOf* y declararla como inversa de *hasIngredient*
- Ejercicio: crear las propiedades *isBaseOf* y *isToppingOf* como subclases de *isIngredientOf*, y establecer sus respectivas propiedades inversas.

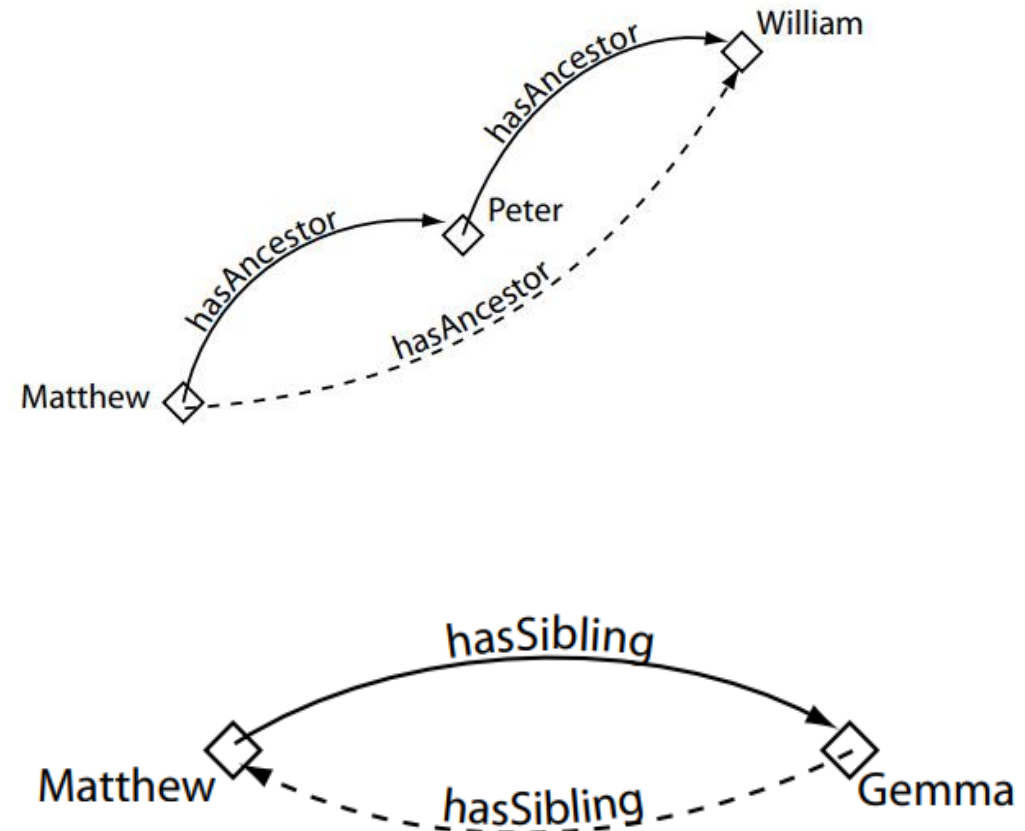
# CARACTERÍSTICAS DE LAS OBJECT PROPERTIES (I)

- **Functional properties:** para una instancia, solamente puede existir una única instancia relacionada a través de dicha propiedad.
- Ejercicio: establecer la propiedad *hasBase* como funcional.
- **Inverse functional properties:** significa que su propiedad inversa es funcional.



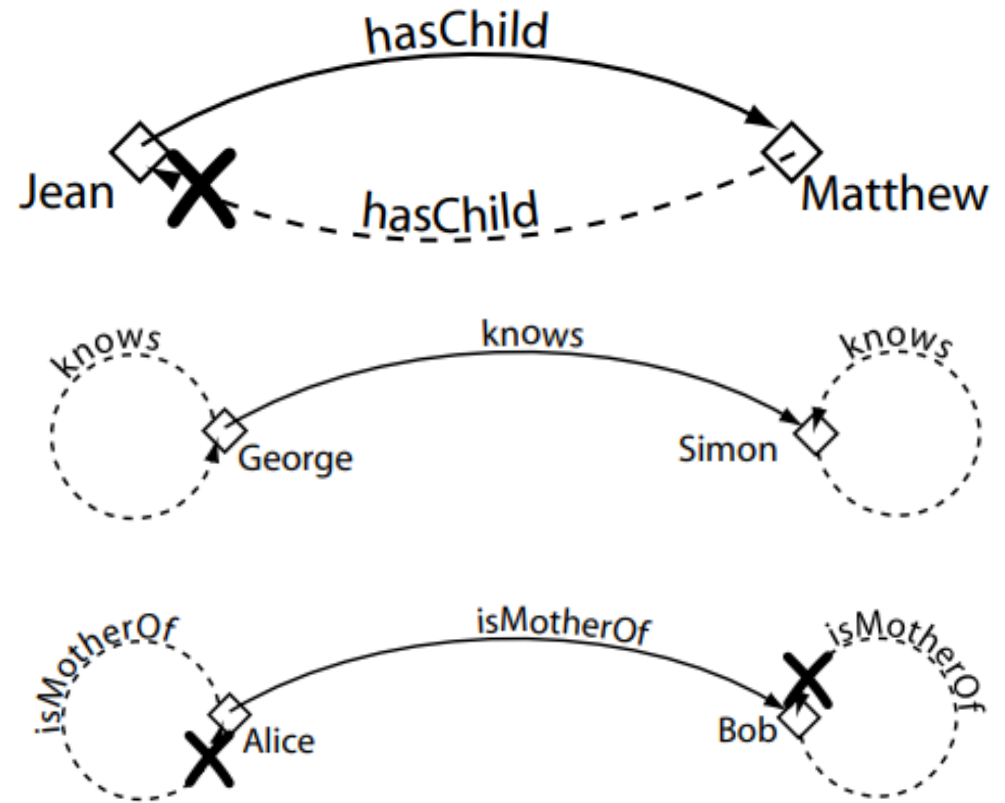
# CARACTERÍSTICAS DE LAS OBJECT PROPERTIES (II)

- **Transitive properties:** si la instancia **a** está relacionada con **b**, y a su vez, **b** con **c**, **a** estará relacionada con **c** a través de la propiedad **P**.
- Ejercicio: establecer la propiedad *hasIngredient* y su inversa como transitivas.
- **Symmetric properties:** si la instancia **a** está relacionada con **b**, entonces **b** también estará relacionada con **a**, a través de la propiedad **P**.



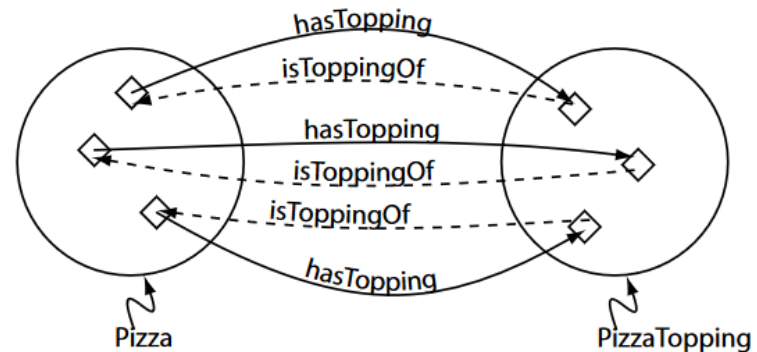
# CARACTERÍSTICAS DE LAS OBJECT PROPERTIES (III)

- **Asymmetric properties:** si la instancia **a** está relacionada con la instancia **b**, **b** no podrá estar relacionada con **a**, a través de la propiedad **P**.
- **Reflexive properties:** esta propiedad tiene que relacionar la instancia **a** consigo misma.
- **Irreflexive properties:** esta propiedad **no** puede relacionar la instancia **a** consigo misma.



# DOMINIOS Y RANGOS DE LAS PROPIEDADES

- Las propiedades deberían tener un **dominio** y un **rango** definidos.
- Una propiedad relaciona instancias del **dominio** con las instancias del **rango**.

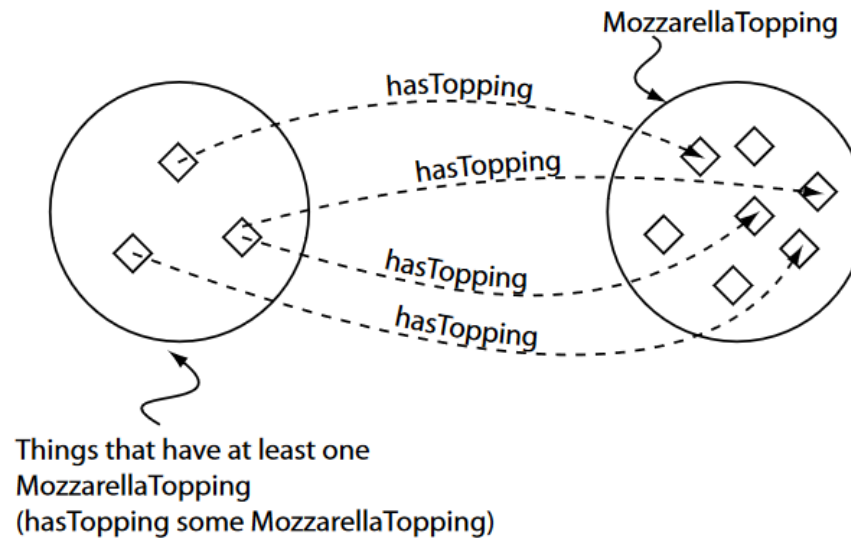


- Ejercicio: establecer *PizzaTopping* como el Rango de la propiedad *hasTopping* y *Pizza* como su Dominio.
- ¿Qué ha pasado con su propiedad inversa?
- Ejercicio: establecer *Pizza* como el Dominio de la propiedad *hasBase* y *PizzaBase* como su Rango.

# DESCRIPCIÓN Y DEFINICIÓN DE CLASES

## PROPERTY RESTRICTIONS

- Una **restricción** describe una clase en base a las relaciones en las que los miembros de dicha clase participan.
- Una restricción es un tipo de clase, igual que una *named class* es un tipo de clase.



# DESCRIPCIÓN Y DEFINICIÓN DE CLASES

## PROPERTY RESTRICTIONS

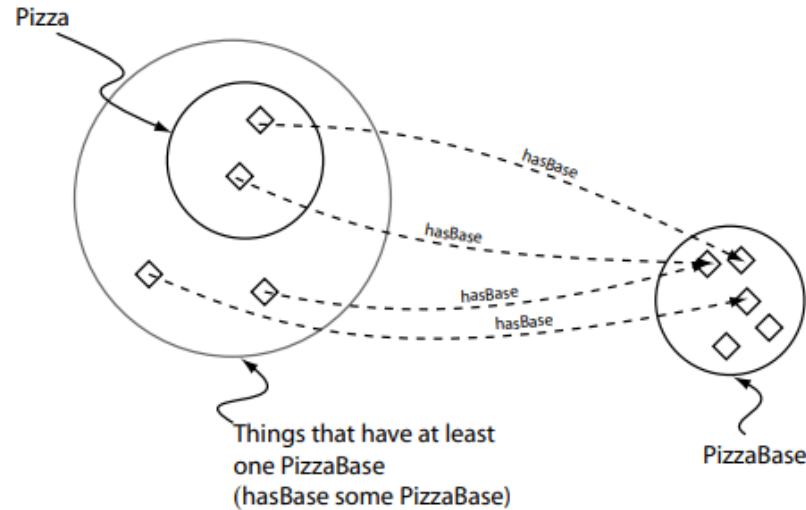
- **Existential restrictions:** describen las clases formadas por aquellas instancias que **al menos participan en una relación** definida por una propiedad determinada y con instancias que pertenecen a una clase determinada.
  - En Protégé se especifica con la palabra **some**.
- **Universal restrictions:** describen las clases formadas por aquellas instancias que para una propiedad específica solamente están relacionadas con las instancias de una clase determinada.
  - En Protégé se especifica con la palabra **only**.
- Una restricción describe una *clase anónima*, que contiene todas las instancias que satisfacen dicha condición.



# DESCRIPCIÓN Y DEFINICIÓN DE CLASES

## PROPERTY RESTRICTIONS

- Ejercicio: añadir una restricción que especifique que toda *Pizza* tiene que tener al menos una *PizzaBase*.



- Las restricciones establecen condiciones *necesarias*, pero no suficientes.

# CREANDO DIFERENTES TIPOS DE PIZZA

- Ejercicio: crear una subclase de *Pizza* denominada *NamedPizza*, la cual a su vez tendrá una subclase denominada *MargheritaPizza*.
- Ejercicio: documentar la recién creada *MargheritaPizza* con el comentario “*A pizza that only has Mozzarella and Tomato toppings*”.
- Ejercicio: crear una restricción que especifique que la *MargheritaPizza* tiene al menos un *MozzarellaTopping*. Hacer lo mismo con *TomatoTopping*.
- Ejercicio: crear la *AmericanPizza*, que está compuesta por Pepperoni, Mozzarella y Tomato.
  - Como es muy parecida a la *MargheritaPizza*, vamos a duplicarla.
- Ejercicio: crear la *AmericanHotPizza*, que es como la *AmericanPizza* con Jalapeños.
- Ejercicio: crear la *SohoPizza*, que es como la *MargheritaPizza* con Olivas y Parmesano.
- Ejercicio: hacer que todas estas pizzas sean disjuntas.

# RAZONAMIENTO

- Principales servicios de un razonador:
  - Comprobar si una clase es subclase de otra.
  - Comprobar la consistencia de la ontología, si en base a los axiomas es posible que se pueda declarar una instancia para cada una de las clases.
    - Una clase será inconsistente si no se puede declarar ninguna clase.
- Ejercicio: crear un *PizzaTopping* denominado *ProbelInconsistentTopping* que sea a la vez subclase de *CheeseTopping* y *VegetableTopping*.
  - Comentar esta clase con la anotación “*This class should be inconsistent when the ontology is classified.*”
  - Ejecutar el razonador.

# CONDICIONES NECESARIAS Y SUFICIENTES

## CLASES PRIMITIVAS Y DEFINIDAS

- Hasta ahora, todas las clases que hemos definido indicaban las condiciones necesarias para pertenecer a dicha clase.
  - Una clase que solamente define condiciones necesarias se denomina **Primitive Class**.
- Ejercicio: crear una *Pizza* denominada *CheesyPizza* con la restricción de que tiene que tener al menos un *CheeseTopping*.
  - Ejecutar el razonador.
- Una clase que define al menos una condición necesaria y una suficiente se denomina **Defined Class**.
- Ejercicio: convertir la clase *CheesyPizza* en una *Defined Class*.

# RESTRICCIONES UNIVERSALES

- Las restricciones existenciales no limitan las clases con las cuales una clase se puede relacionar a través de una propiedad determinada.
- Ejercicio: observar las restricciones de *AmericanPizza*.
- Las restricciones universales no especifican la necesidad de que una relación exista, simplemente especifican que, de existir, tiene que ser con las instancias de una clase determinada.
- Ejercicio: crear una clase *VegetarianPizza* que **solamente** pueda tener *CheeseTopping* o *VegetableTopping*.
  - Ejecutar el razonador.
- Ejercicio: convertir la clase *VegetarianPizza* en *Defined Class*.
  - Ejecutar el razonador.

# CLASIFICACIÓN AUTOMÁTICA Y MUNDO ABIERTO

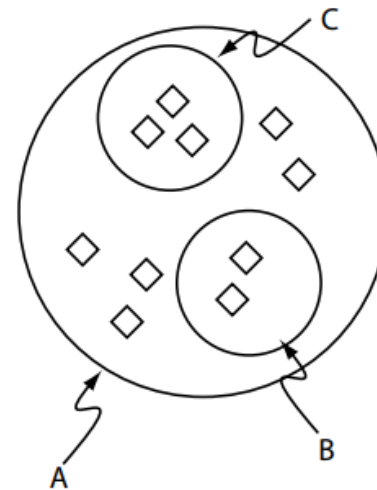
- Al definir las diferentes pizzas, en ningún momento hemos dicho que esos sean sus únicos ingredientes, solamente que **al menos** tienen que tener esos ingredientes.
  - Por lo tanto, el clasificador no las puede clasificar como *VegetarianPizza*.
- **Closure axioms:** una restricción universal que establece que una propiedad solamente puede relacionar instancias de una determinada clase.
  - Esta restricción será la unión de todas las clases que ocurran en las restricciones existenciales.
- Ejercicio: establecer el *Closure axiom* para la *MargheritaPizza*.
  - Ejecutar el razonador.
- Ejercicio: establecer *Closure axioms* para el resto de las subclases de *NamedPizzas*.

# VALUE PARTITIONS

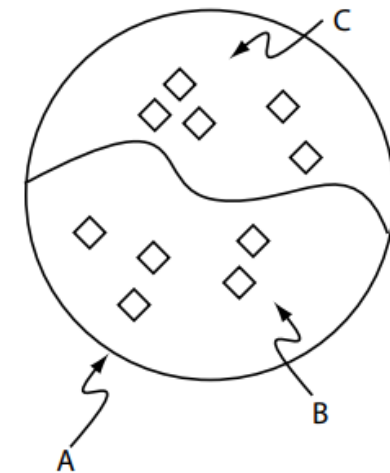
- Las *Value Partitions* son un patrón de diseño.
  - Diseñados por expertos para dar respuesta a problemas de diseño recurrentes.
  - Especificado en <https://www.w3.org/TR/2005/NOTE-swbp-specified-values-20050517/>
- Vamos a crear una *Value Partition* para especificar el *spiciness* de las pizzas.
- Ejercicio:
  - Crear la subclase de *owl:Thing ValuePartition*, que a su vez tendrá a *SpicinessValuePartition* como subclase.
  - Crear las clases *Hot*, *Mild* y *Medium* como subclases de *SpicinessValuePartition*, y que sean disjuntas entre sí.
  - Crear la *object property hasSpiciness*, y establecer *SpicinessValuePartition* como el rango de esta propiedad.

# COVERING AXIOMS

- Están formados por una clase “solapada” y por las clases que “solapan”.
- Supongamos que las clases **B** y **C** son subclases de **A**.
- Tenemos un *covering axiom* que establece que **A** es solapada por **B** y **C**.
  - Esto implica que cualquier instancia de **A** tendrá que pertenecer también a **B** y/o **C**.
- Si **B** y **C** son disjuntas, cualquier instancia de **A** tendrá que pertenecer o bien a **B**, o bien a **C**.



Without a covering axiom  
(B and C are subclasses of A)

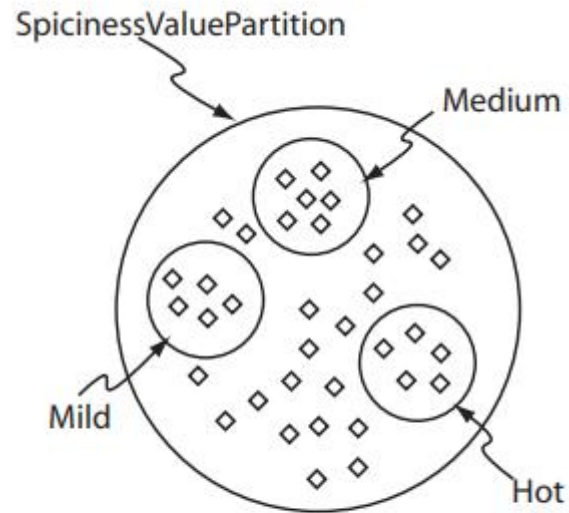


With a covering axiom  
(B and C are subclasses of A  
and A is a subclass of B union C)

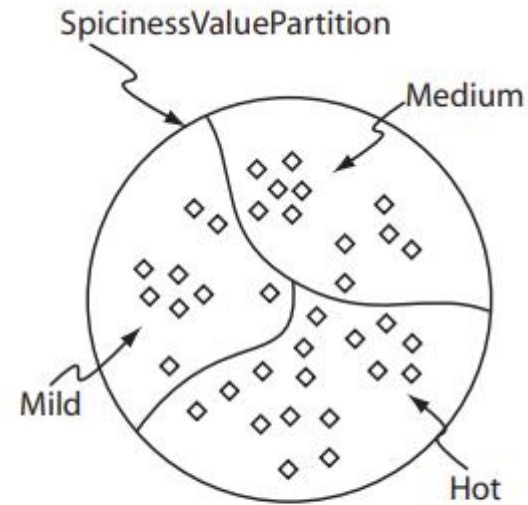


# COVERING AXIOMS

- Ejercicio: añadir un *covering axiom* para la clase *SpicinessValuePartition*.



Without a covering axiom



With a covering axiom  
(SpicinessValuePartition is covered by  
Mild, Medium, Hot)

# AÑADIENDO SPICINESS A LOS PIZZATOPPINGS

- Añadir la restricción “*hasSpiciness some Hot*” a *JalapenoTopping*.
- Añadir restricciones similares a todas las subclases de *PizzaTopping*.
  - ¿Uno a uno? Podemos utilizar el *Matrix Plugin*.
- Acceder a *File* → *Check for plugins* e instalar *Matrix View*.
  - Reiniciar Protégé.
- Activamos la vista en *Window* → *Tabs* → *Class Matrix* y accedemos a ella.
- Arrastramos la propiedad *hasSpiciness* al tablero central.
- Escribir el valor de la propiedad o arrastrar desde el panel *Classes Palette*.
- Crear las *Defined Classes* *MidSpicyPizza* y *NonSpicyPizza*.
  - Ejecutar el razonador.

# RESTRICCIONES DE CARDINALIDAD

- Restricciones que permiten definir el mínimo, máximo o relaciones exactas que puede tener una instancia a través de una propiedad determinada.
- Ejercicio: crear una subclase de *Pizza* denominada *InterestingPizza* que tenga **al menos 3 *PizzaToppings*** y convertirla en *Defined Class*.
  - Ejecutar el razonador.
- ***Qualified Cardinality Restrictions***: además de indicar cuántas relaciones, indica la clase a la cual tienen que pertenecer las instancias relacionadas.
- Ejercicio: crear una subclase de *Pizza* denominada *FourCheesePizza* que tenga **exactamente 4 *PizzaToppings***.

# DATATYPE PROPERTIES

- Relacionan una instancia con un valor de uno de los tipos de **XLM Schema** o con un **literal RDF**.
- Ejercicio: crear un *Datatype Property* denominado *hasCalorificContentValue*.
- Ejercicio: crear las instancias *Example-Margherita* con el valor 263 para la propiedad *hasCalorificContentValue*, y *QuattroFormaggio* con el valor 723 para la misma propiedad.
- Estas propiedades también tienen restricciones.
- Ejercicio: crear una restricción que establezca que todas las instancias de la clase *Pizza* tienen que tener una relación *hasCalorificContentValue* con un valor *integer*.

# DATATYPE PROPERTIES

- Ejercicio: crear una subclase de *Pizza* denominada *HighCaloriePizza* definida como “cualquier pizza que tenga un valor calórico superior a 400” y convertirla a *Defined Class*.
- Ejercicio: hacer lo propio con *LowCaloriePizza*.
- Ejercicio: establecer la propiedad *hasCalorificContentValue* como *functional*.

# CUESTIONES SOBRE EL MUNDO ABIERTO

- Ejercicio: crear la subclase de *Pizza* denominada *NonVegetarianPizza*, que incluya todas las pizzas no incluidas por *VegetarianPizza*.
  - Para ello, vamos a definir *NonVegetarianPizza* como **complementaria** de *VegetarianPizza*.
- Ejercicio: crear una subclase de *NamedPizza* denominada *UnclosedPizza* que establezca que al menos tiene que tener un *MozzarellaTopping*.

# CREACIÓN DE INSTANCIAS

- Vamos a crear países para determinar el origen de las diferentes pizzas.
- No sería correcto crear una clase “*England*”, ya que no tendría sentido declarar “cosas que son instancias de *England*”.
  - Lo lógico sería tener una clase *Country* e instancias de la misma.
- Ejercicio: crear una clase *Country* y crear las instancias *England*, *Italy*, *USA*, *France* y *Germany*.
  - Indicar que son instancias distintas.
- Ejercicio: crear la object property *hasCountryOfOrigin*
- Ejercicio: establecer *Italy* como el país de origen de *MozzarellaTopping*.

# ENUMERATED CLASSES

- OWL nos permite definir una clase listando de manera específica las instancias que la forman.
- Por ejemplo, podríamos tener una clase *DaysOfTheWeek* que estuviese descrita por las instancias e {*Sunday Monday Tuesday Wednesday Thursday Friday Saturday*} (y solo esas).
- Ejercicio: convertir la clase *Country* en una clase enumerada formada por *England, Italy, USA, France y Germany*.



# ANNOTATION PROPERTIES

- OWL permite anotar las clases, propiedades, etc.
- Es una buena práctica para que la ontología se entienda y sea reutilizable.
- Las anotaciones siempre tienen que relacionar la clase/propiedad/etc. con un literal.
- No pueden tener jerarquía como las clases y propiedades.
- Las más típicas son *rdfs:label* y *rdfs:comment*.