# OWL

# Web Ontology Language

# RDF & RDFS

- RDF does not assign any specific meaning to the elements of a triple
  - Interpretation is an arbitrary binary relation
- RDF Schema allows one to define vocabulary terms and the relations between those terms
  - it gives explicit meaning (semantics) to particular RDF predicates and resources
  - it specifies how elements of an RDF triplet should be interpreted

# Issues with RDFS

- RDFS is too weak to describe resources in sufficient detail; in particular, it does *not* have:
  - *localised range and domain* constraints
    - E.g., one cannot say that the range of *hasChild* is person when applied to people and cat when applied to cats
  - *existence/cardinality* constraints
    - E.g., one cannot say that all instances of the Person class have a mother that is also a Person, or that a Person has exactly 2 parents
  - *transitive, inverse or symmetrical* properties
    - E.g., one cannot say that *isPartOf* is a transitive property, that *hasPart* is the inverse of *isPartOf*, or that *touches* is symmetrical

# Requirements for a Web Ontology Language

Desirable features of a Web Ontology Language:

- Extends existing Web standards
  - Such as XML, RDF, RDFS
- Easy to understand and use
- Formally specified
- Of “adequate” expressive power
- Enables automated reasoning

# OWL (Web Ontology Language) through examples

# Intro to OWL: Example 1

- Let's start by defining some concepts and their relationship
  - define the concepts Camera and SLR
  - state that SLR is a specific kind of Camera

```
c:Camera rdf:type owl:Class .
```

```
c:SLR rdf:type owl:Class ;
        rdfs:subClassOf c:Camera .
```

# Intro to OWL: Example 1

- This definition helps the computer to differentiate between SLR as a kind of camera (Single Lens Reflex) and all other possible meanings of this term

SLR **Single Lens Reflex** (camera)
SLR Satellite Laser Ranging
SLR Self Loading Rifle
SLR Sending Loudness Rating (telecommunications)
SLR Service Level Report
SLR Service Location Register
SLR Side Looking Radar
SLR Single Line Restoral
SLR Single Linear Recording
SLR Slide Raft (aircraft door)
SLR Slush on Runway(s)
SLR Solectron
SLR Spacelift Range
SLR Sri Lanka Rupee (national currency)
SLR Statutory Liquidity Ratio
SLR Stock Level Report
SLR Stock Level Requirement
SLR Straight Leg Raise
SLR Straight Leg Raising
SLR System Level Requirement(s)

# Intro to OWL: Example 1

- Scenario
  - Tom is interested in purchasing a *camera* with the following features:
    - 75-300mm *zoom lens*,
    - *aperture* of 4.5-5.6,
    - *shutter speed* that ranges from 1/500 sec. to 1.0 sec
  - Tom issues a request to his personal Web bot to crawl the Web looking for Web sites offering the camera he wants

# Intro to OWL: Example 1

- Scenario (cont.)
  - Let's assume that there exists an OWL-based Camera Ontology, which the Web Bot can "consult" while it travels across the Web

# Intro to OWL: Example 1

- Scenario (cont.)
  - the Web Bot finds this document at a Web site:

Is it relevant?
(SLR = Single Lens Reflex)

```
<PhotographyStore rdf:ID="Hunts"
                  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <store-location>Malden, MA</store-location>
  <phone>617-555-1234</phone>
  <catalog rdf:parseType="Collection">
    <SLR rdf:ID="Olympus-OM-10"
         xmlns="http://www.camera.org#">
      <lens>
        <Lens>
          <focal-length>75-300mm zoom</focal-length>
          <f-stop>4.5-5.6</f-stop>
        </Lens>
      </lens>
      <body>
        <Body>
          <shutter-speed rdf:parseType="Resource">
            <min>0.002</min>
            <max>1.0</max>
            <units>seconds</units>
          </shutter-speed>
        </Body>
      </body>
      <cost rdf:parseType="Resource">
        <rdf:value>325</rdf:value>
        <currency>USD</currency>
      </cost>
    </SLR>
  </catalog>
</PhotographyStore>
```

# Intro to OWL: Example 1

- Scenario (cont.)
  - the Web Bot finds this document at a Web site:

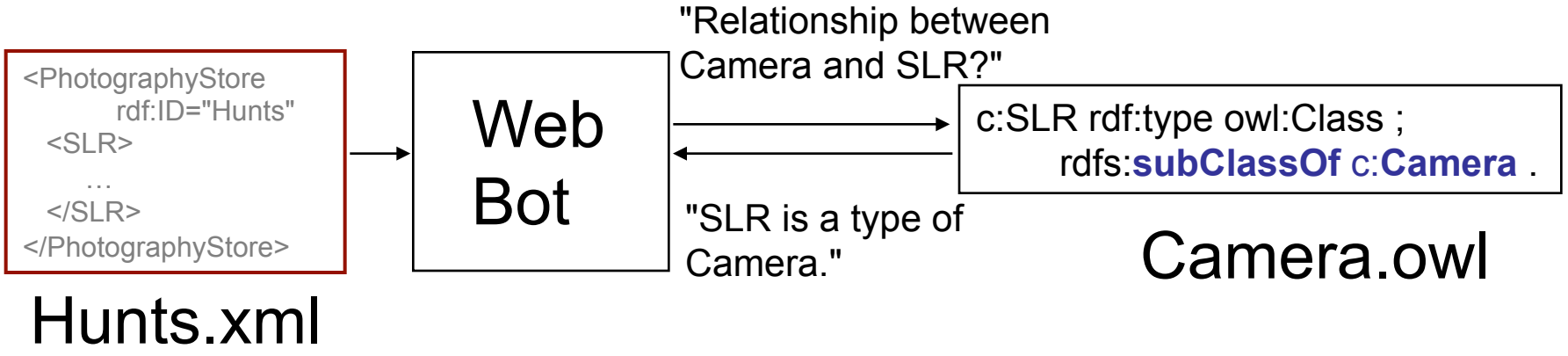
```
<Camera xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.camera.org#">
  <lens>
    <Lens>
      <size>75-300mm zoom</size>
      <aperture>4.5-5.6</aperture>
    </Lens>
  </lens>
  <body>
    <Body>
      <shutter-speed rdf:parseType="Resource">
        <min>0.002</min>
        <max>1.0</max>
        <units>seconds</units>
      </shutter-speed>
    </Body>
  </body>
</Camera>
```

Match?

```
<PhotographyStore rdf:ID="Hunts"
                  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <store-location>Malden, MA</store-location>
  <phone>617-555-1234</phone>
  <catalog rdf:parseType="Collection">
    <SLR rdf:ID="Olympus-OM-10"
         xmlns="http://www.camera.org#">
      <lens>
        <Lens>
          <focal-length>75-300mm zoom</focal-length>
          <f-stop>4.5-5.6</f-stop>
        </Lens>
      </lens>
      <body>
        <Body>
          <shutter-speed rdf:parseType="Resource">
            <min>0.002</min>
            <max>1.0</max>
            <units>seconds</units>
          </shutter-speed>
        </Body>
      </body>
      <cost rdf:parseType="Resource">
        <rdf:value>325</rdf:value>
        <currency>USD</currency>
      </cost>
    </SLR>
  </catalog>
</PhotographyStore>
```

# Intro to OWL: Example 1

- Scenario (cont.)
  - the Web Bot "consults" the OWL Camera Ontology



# Intro to OWL: Example 1

- Scenario (cont.)
  - the Web Bot "consults"
    the OWL Camera Ontology

```
c:focal-length rdf:type owl:DatatypeProperty ;
        owl:equivalentProperty c:size ;
        rdfs:domain c:Lens ;
        rdfs:range xsd:string .
```

Interpretation:
focal-length has the same meaning as (lens) size;
it should be used with the Lens class;
it's value should be of the type xsd:string.

# Intro to OWL: Example 1

- Scenario (cont.)
  - the Web Bot "consults"
    the OWL Camera Ontology

```
c:f-stop rdf:type owl:DatatypeProperty ;
        owl:equivalentProperty c:aperture"/>
        rdfs:domain c:Lens ;
        rdfs:range rdf:resource xsd:string .
```

"f-stop is equivalent to aperture."

# Example 1: Lessons Learned

- Summary:
  *interoperability despite terminological differences*!
  - achieved through the use of the OWL Camera Ontology
  - allows for overcoming situations where different terms, or even vocabularies are used to describe the same thing (resource or relationship)
    - E.g., VCard and FOAF for describing people, Schema.org and GoodRelations for describing products

# Example 2: The Robber and the Speeder

Scenario:

DNA samples from a robbery identified

Michael Smith as the suspect.

Here is the police report on the robbery:

```
<report-2003-03-17-XTf4> rdf:type p:Robbery ;
    dcterms:description “...” ;
    p:suspect <http://www.dna-bank.org/people/MichaelSmith> .

<http://www.dna-bank.org/people/MichaelSmith>
                                                rdf:type foaf:Person .
```

# Example 2: The Robber and the Speeder

## Scenario (cont.):

Later in the day a state trooper gives a person a ticket for speeding. The driver's license showed the name Mike. Here is the state trooper's report on the speeder:

```
<report-2003-03-17-QWRP>  rdf:type p:Speeder ;
    dcterms:description “...” ;
    p:driver <http://www.nypd.gov/speeders/MikeSmith> .

<http://www.nypd.gov/speeders/MikeSmith>
                                        rdf:type foaf:Person ;
                                        foaf:name “Mike” .
```

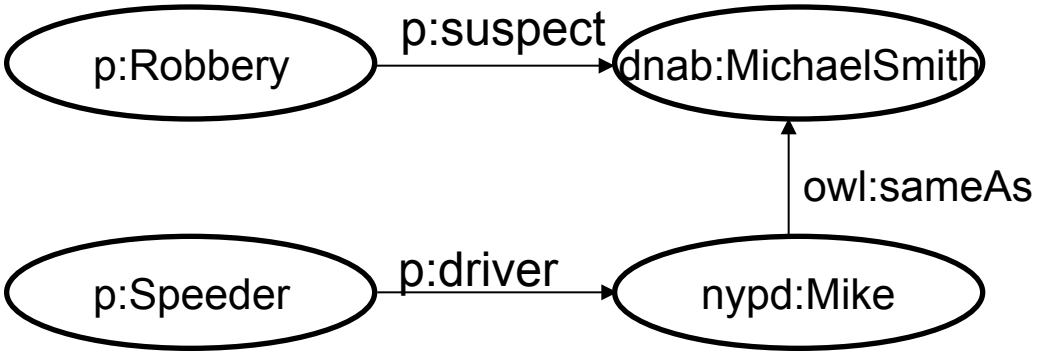
# Example 2: The Robber and the Speeder

Any relationship between
the Robber and the Speeder?

# Example 2: The Robber and the Speeder

The Central Intelligence Agency (CIA) has a file on Mike:

<http://www.nypd.gov/speeders/MikeSmith> rdf:type foaf:Person ;
**owl:sameAs** <http://www.dna-bank.org/people/MichaelSmith>

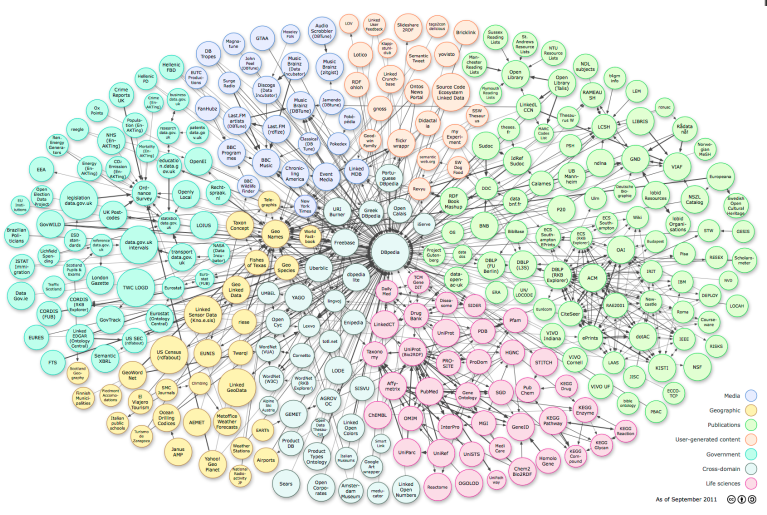



The local police, state troopers, and CIA share their information, thus enabling the following inference to be made:

**Inference**: The Robber and the Speeder are one and the same!

# Example 2: Lesson Learned

- OWL provides the *owl:sameAs* property allowing us to state that two URIs refer to the same resource
- *owl:sameAs* property is by far the most frequently used for linking data from different datasets of the Linked Data Cloud



Check: [http://sameas.org/](http://sameas.org/)

# Example 3:
# The Birthplace of Nicole Kidman is …

## Scenario:

Upon scanning the Web, a Web bot has found 3 documents that contain data about Nicole Kidman:

1
```
<http://www.celebreties.org#Nicole_Kidman> a schema:Person ;
        schema:birthPlace <http://www.states.org#Hawaii>
```

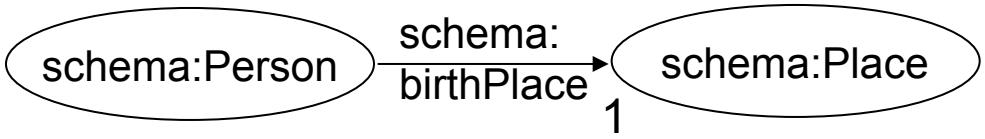
2
```
<http://www.celebreties.org#Nicole_Kidman> a schema:Person ;
        schema:birthPlace <http://www.history.org#Sandwich_Islands>
```

3
```
<http://www.celebreties.org#Nicole_Kidman> a schema:Person ;
        schema:birthPlace <http://www.tourism.org#Aloha_State>
```

**Question**: What is the birthplace of Nicole Kidman?

# Example 3:
# The Birthplace of Nicole Kidman is …

The OWL version of Schema.org states that a Person can have only one birthplace location:




```
schema:birthPlace rdf:type owl:ObjectProperty ;
        rdf:type owl:FunctionalProperty ;
        rdfs:domain schema:Person ;
        rdfs:range schema:Place .
```

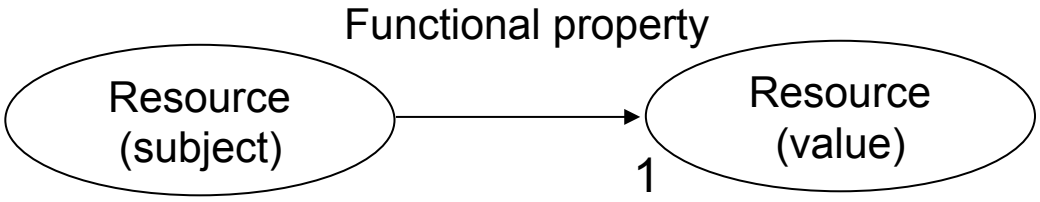
Thus, the ontology enables the following inference to be made:

<http://www.states.org#Hawaii>,
<http://www.history.org#Sandwich_Islands>, i
<http://www.tourism.org#Aloha_State>

**all represent the same location!**

# Example 3: Lesson Learned

This example illustrates the OWL's feature that allows us to specify that the subject resource has exactly one value for a specific property:




Examples of properties of this type are birth date, social security number, vehicle registration number, and the like

# Example 3: Lesson Learned

- The example also illustrates how we can use OWL to identify cases when multiple URIs refer to the same resource;
- It allows for resolving issues that arise from the practice that different people and organizations tend to generate their own URI for a resource that already has a URI

# More OWL features (through examples)

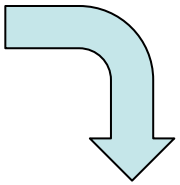
# Inverse property

Let's suppose that the *ex:hasAdvisor* property is defined as an inverse property of *ex:hasPhDStudent*

```
ex:hasAdvisor rdf:type owl:ObjectProperty ;
        owl:inverseOf  ex:hasPhDStudent ;
        rdfs:domain ex:Person ; rdfs:range ex:Person .
```

Let's further suppose that an RDF graph contains the following triplets:

```
…
ex:Jim rdf:type ex:Person ;
        ex:hasAdvisor ex:John .
…
```



Facts that can be inferred about ex:John

```
ex:John
        ex:hasPhDStudent ex:Jim ;
        rdf:type ex:Person .
```

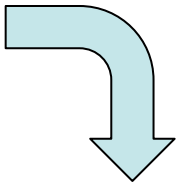
# Transitive property

Let's suppose that *ex:locatedIn* is defined as an owl:transitiveProperty

```
ex:locatedIn rdf:type owl:ObjectProperty , owl:transitiveProperty ;
             rdfs:domain ex:Place ; rdfs:range ex:Place .
```

Let's further suppose that an RDF graph contains the following triplets:

```
…
ex:Bath ex:locatedIn ex:England .
…
ex:England
    ex:locatedIn ex:GreatBritain .
…
```



Facts that can be inferred about ex:Bath

```
…
ex:Bath rdf:type ex:Place ;
        ex:locatedIn ex:GreatBritain .
…
```

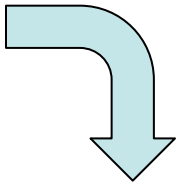
# Symmetric property

Let's suppose that *ex:friendOf* is defined as an owl:symmetricProperty

```
ex:friendOf a owl:ObjectProperty , owl:symmetricProperty ;
            rdfs:domain ex:Person ; rdfs:range ex:Person .
```

Let's further suppose that an RDF graph contains the following triplet about an unknown resource ex:Philippe:

```
…
ex:Jack
        ex:friendOf ex:Philippe .
…
```



Facts that can be inferred about ex:Philippe

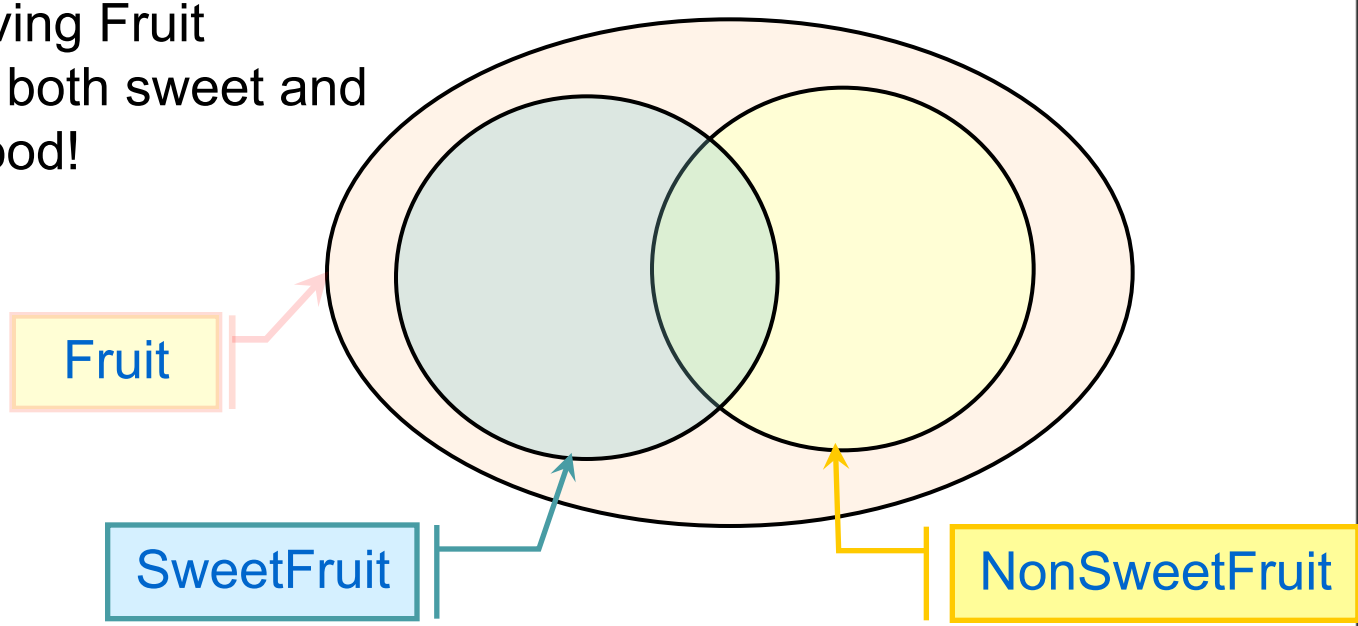
```
ex:Philippe
            a ex:Person ;
            ex:friendOf ex:Jack .
```

# Disjoint classes

Let's suppose that an OWL ontology defines the following 3 classes

```
ex:Fruit a owl:Class .
ex:SweetFruit a owl:Class ; rdfs:subClassOf ex:Fruit .
ex:NonSweetFruit a owl:Class ; rdfs:subClassOf ex:Fruit .
```

This allows for having Fruit instances that are both sweet and not sweet – not good!

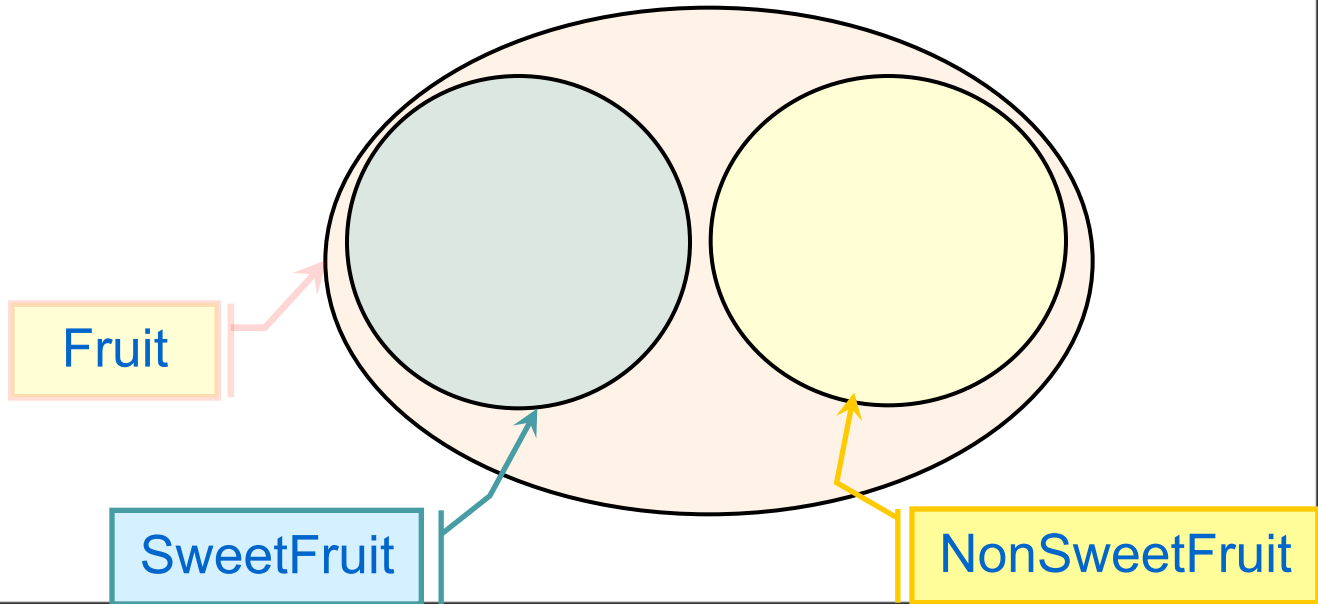



**Important**: classes in OWL are defined as *sets of individuals*

# Disjoint classes

To state that two classes cannot have any instance in common, we have to declare them as *disjoint classes*

```
ex:Fruit a owl:Class .
ex:SweetFruit a owl:Class ; rdfs:subClassOf ex:Fruit .
ex:NonSweetFruit a owl:Class ;
        rdfs:subClassOf ex:Fruit ; owl:disjointWith ex:SweetFruit .
```



# Restrictions in OWL: existential restriction

Let's suppose the following is a small excerpt of an OWL ontology for modeling houses, their rooms and furniture

```
ex:Room a owl:Class .

ex:Furniture a owl:Class .

ex:furnishedWith a owl:ObjectProperty ;
        rdfs:domain ex:Room ; rdfs:range ex:Furniture .

ex:Bedroom a owl:Class ; rdfs:subClassOf ex:Room .

ex:Bed a owl:Class ; rdfs:subClassOf ex:Furniture .
```

Now, we want to state that bedroom is a kind of room that has to have at least one bed.

To do that, we can use *owl:someValuesFrom* restriction

# Restrictions in OWL: existential restriction

```
ex:Bedroom a owl:Class ;
        rdfs:subClassOf ex:Room ;
        rdfs:subClassOf [
                a owl:Restriction ;
                owl:onProperty ex:furnishedWith ;
                owl:someValuesFrom ex:Bed
        ] .
```

Interpretation:

Bedroom is a kind of Room that can have different pieces of Furniture, but at least one piece has to be a Bed.

# Restrictions in OWL: universal restriction

Let's suppose the following is a small excerpt of an OWL ontology for modeling food and dishes

```
ex:Dish a owl:Class .

ex:Ingredient a owl:Class .

ex:hasIngredient a owl:ObjectProperty ;
        rdfs:domain ex:Dish ; rdfs:range ex:Ingredient .

ex:VegetarianDish a owl:Class ; rdfs:subClassOf ex:Dish .

ex:VegetarianIngredient a owl:Class ;
                rdfs:subClassOf ex:Ingredient .
```

Now, we want to state that a VegetarianDish has to have only VegetarianIngredients.

To do that, we can use *owl:allValuesFrom* restriction

# Restrictions in OWL: existential restriction

```
ex:VegetarianDish a owl:Class ;
        rdfs:subClassOf ex:Dish ;
        rdfs:subClassOf [
                a owl:Restriction ;
                owl:onProperty ex:hasIngredient ;
                owl:allValuesFrom ex:VegetarianIngredient
        ] .
```

Interpretation:

VegetarianDish is a kind of Dish that can consist only of VegetarianIngredients.

# Restrictions in OWL: value restriction

Let's suppose the following is a small excerpt of an OWL ontology for modeling wine

```
ex:Wine a owl:Class .

ex:color owl:DatatypeProperty ;

        rdfs:domain ex:Wine ; rdfs:range xsd:string .
```

Now, we want to introduce class RedWine as a kind of Wine that has the red color.

To do that, we can use *owl:hasValue* restriction

# Restrictions in OWL: value restriction

```
ex:RedWine a owl:Class ;
        rdfs:subClassOf ex:Wine ;
        rdfs:subClassOf [
                a owl:Restriction ;
                owl:onProperty ex:color ;
                owl:hasValue “red”^^xsd:string
        ] .
```

Interpretation:

RedWine is a kind of Wine that has the red color .

The Wine ontology has for long been the most widely used ontology for teaching/learning OWL

# Using set operators to create a class

Since classes in OWL are, in fact, sets of individuals, classes can be also defined using set operators:

union, intersection, complement

Examples:

```
ex:Mother  owl:equivalentClass  [
          a  owl:Class ;
          owl:intersectionOf  ( ex:Woman ex:Parent )
 ] .
```

```
ex:ChildlessPerson  owl:equivalentClass  [
          a  owl:Class ;
          owl:intersectionOf  ( ex:Person
                                [ owl:complementOf  :Parent ] )
 ] .
```

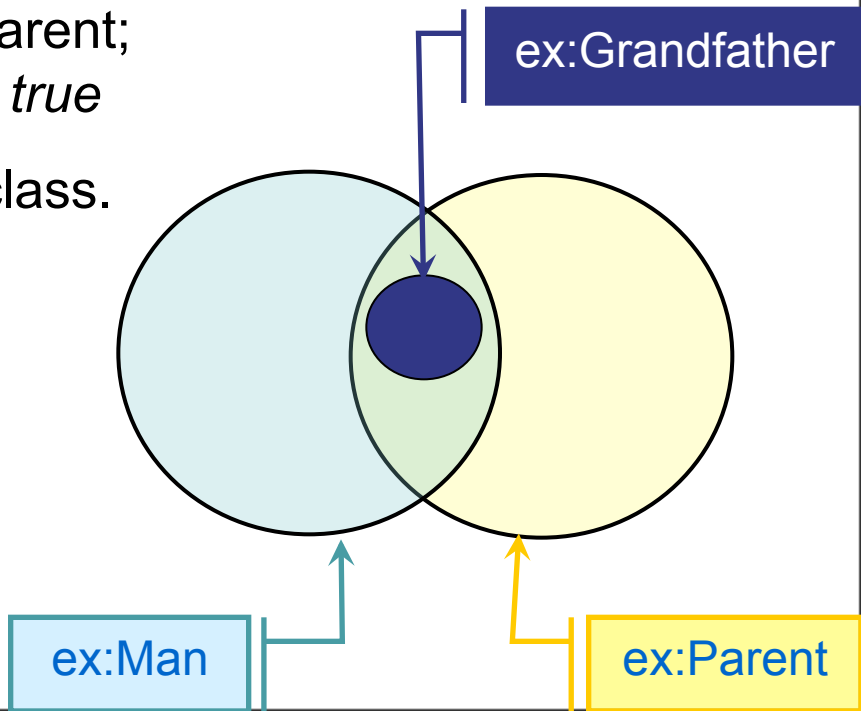
# Difference between *described* and *defined* classes

```
ex:Grandfather  rdfs:subClassOf  [
            a   owl:Class ;
            owl:intersectionOf  ( ex:Man  ex:Parent )
 ]
```

Interpretation:

every Grandfather is both a man and a parent;
however, the *converse is not necessarily true*

We say that Grandfather is a *described* class.



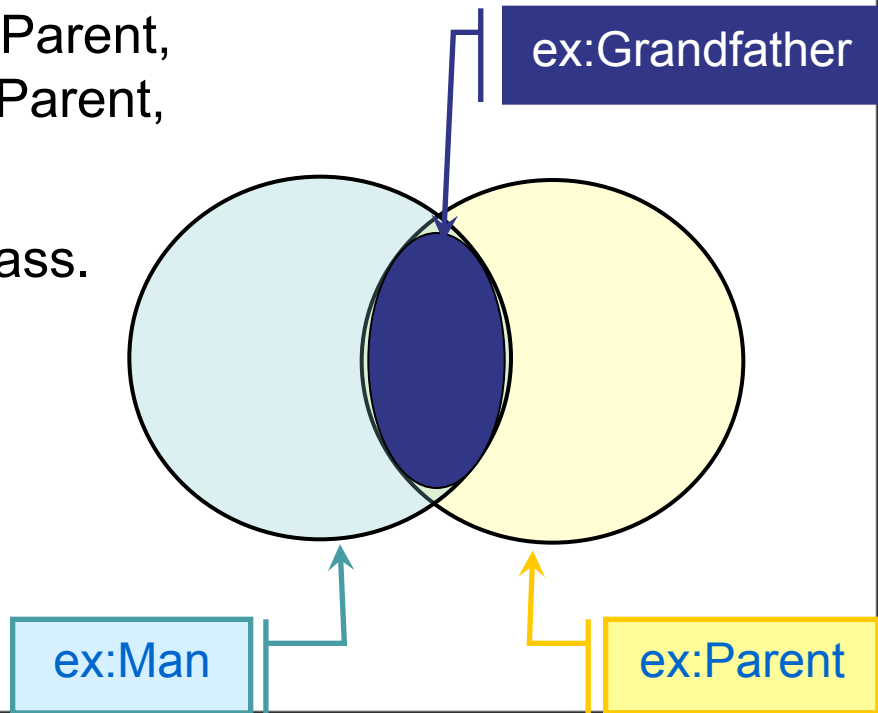
# Difference between *described* and *defined* classes

```
ex:Grandfather  rdfs:equivalentClass  [
            a   owl:Class ;
            owl:intersectionOf  ( ex:Man  ex:Parent )
 ]
```

Interpretation:

every Grandfather is both a Man and a Parent,
and if a resources is both a Man and a Parent,
it is also a Grandfather

We say that Grandfather is a *defined* class.



# Learn more about OWL

OWL offers many more features for knowledge modeling

These are well described in OWL 2 Primer:

http://www.w3.org/TR/2009/WD-owl2-primer-20090611/

Also, highly useful could be the following OWL tutorials:

- TopBraid Composer Getting Started Guide and OWL Tutorial (link)
  - OWL tutorial + step-by-step introduction into TBC modeling environment
- Protege OWL Tutorial (link)
  - OWL tutorial + step-by-step introduction into the Protégé modeling environment

# Some well known publicly available OWL ontologies

- Financial Industry Business Ontology (FIBO): http://www.edmcouncil.org/financialbusiness
- Gene ontology: http://geneontology.org/
- GeoNames ontology: http://www.geonames.org/ontology/
- BBC ontologies: http://www.bbc.co.uk/ontologies
- OWL version of Schema.org: http://topbraid.org/schema/