Simone Preite 3885035

Question 1.

Write a small ontology to describe the zoo domain (e.g., tigers, monkeys, penguins, visitors, tickets, snacks etc.). Introduce some classes, properties, and individuals. Use some of the RDFS and OWL vocabulary introduced in the lecture.

dbpedia: <http://dbpedia.org/ontology>

ex:Zoo	rdf:type	owl:Class;
ex:Zoo	owl:sameAs	dbpedia:Zoo;
ex:Snack	rdfs:subClassOf	ex:Zoo;
ex:Animal	rds:type	owl:Class;
ex:Animail	owl:sameAs	owl:Zoo;
ex:Tiger	rdfs:type	ex:Animal;
ex:Monkey	rdfs:type	ex:Animal;
ex:Pengion	rdfs:type	ex:Animal;
ex:buys	rdf:type	owl:ObjextProperty;
	rdf:domain	dbpedia:person
	rdf:range	ex:Snack

Question 2.

Use OWL to model the following axioms.

1. Pizzas always have at least two toppings.
2. Every Margherita pizza has a tomato topping.
3. No Margherita pizza has a meat topping.

1.

```
ex:Pizza rdf:type owl:Restriction;
        owl:onProperty ex:topping;
        owl:minCardinality "2"^^xsd:nonNegativeInteger .
```

2.

```
ex:MargheritaPizza rdfs:subClassOf [
        rdf:type owl:Restriction;
        owl:onProperty ex:topping;
        owl:hasValueFrom ex:Tomato
].
```

3.

```
NegativeObjectPropertyAssertion(ex:Topping ex:MargheritaPizza ex:Meat).
```

Question 3.

Give an example of a property that is:
1. Transitive
2. Symmetric
3. Functional
4. Inverse Functional

Be creative – do not list those already presented in the lecture.

```
1. ex:hasBrother          rdf:type        owl:TransitiveProperty.
2. ex:hasSiblings         rdf:type        owl:SymmetricProperty.
3. ex:hasSocialSecurityNumber     rdf:type        owl:FunctionalProperty.
4. ex:isSocialSecurityNumber      rdf:type        owl:InverseFunctionalProperty.
```

Question 4.

In OWL Lite, the property owl:disjointWith is not allowed.
Given two classes A and B, find a way to state in OWL Lite that these two classes are disjoint.

```
ex:A    rdf:type        owl:Class.
ex:B    rdf:type        owl:Class.
Ex:C    rdf:type        [<owl:Class    rdf:ID=”ClassName”>
                            <owl:intersectionOf   rdf:parseType=”Collection”>
                                <owl:Class     rdf:about=”A”/>
                                <owl:Class     rdf:about=”B”/>
                            </owl:intersectionOf>
                         <owl:Class>]
```

Question 5.

Imagine a game with the name Stones. It is played by two players where players take turns – each turn is another step in the game.
An experienced player is a player that participated in at least two games of stones where in each game a special situation occurred. A special situation is when exactly one red stone and one blue stone is won as the result of a single move.
Use OWL restriction classes to model a class ex:ExperiencedPlayer that contains all experienced players. Via reasoning it should be possible that player that satisfy the conditions of being experienced players are automatically assigned to the class ex:ExperiencedPlayer. Use Turtle syntax to describe your solution.
Hint: You might need multiple restriction classes.

```
ex: Stones  rdf:subClassOf  [
 rdf:type   owl:Restriction ;
 owl:onProperty   ex:turn ;
 owl:minCardinality    “1”^^xsd:nonNegativeInteger .

ex: Stones  rdf:subClassOf  [
 rdf:type   owl:Restriction ;
 owl:onProperty  ex:player ;
 owl:Cardinality  “2”^^xsd:nonNegativeInteger .

ex:Special_situation  owl:unionof  (
ex:Stone [
  rdf:type   owl:Restriction ;
  owl:onProperty  ex:color ;
  owl:hasValue   ex:Blue ;
  owl: Cardinality  “1”^^xsd:nonNegativeInteger .
 ]
ex:Stone  [
  rdf:type   owl:Restriction ;
  owl:onProperty  ex:color ;
  owl:hasValue   ex:Red ;
  owl: Cardinality  “1”^^xsd:nonNegativeInteger .
 ]
ex:Stones  [
  rdf:type   owl:Restriction ;
  owl:onProperty  ex:turn ;
  owl:Cardinality  “1”^^xsd:nonNegativeInteger
 ] ) .

ex:experiencedPlayer  rdf:type     owl:Restriction ;
    owl:onProperty   ex:special_situation ;
```

owl:Cardinality “2”^^xsd.nonNegativeInteger .