Kashi Vishwanath Bondugula Submission

1. We have a base class of type Provider
2. Then we create a class of type Surgeon whose base is Provider
3. Radiologist is also a type of Provider, hence we create a subclass from it
4. Same goes with the Internist
5. For testing purposes, I have created one surgeon using rdf:description and rdf:type as Surgeon, Internist, Radiologist
6. Sam is a surgeon in my code, John is a Radiologist, Charlotte is a provider and intern is an Internist.
7. I have tested if they exist or not using elems command
    a. elems Surgeon
    b. elems Radiologist
    c. elems Internist
    d. elems Provider
8. Moving on we have Treatment base class
9. Surgery, Radiology and DrugTreatment are different forms of treatment made using subClassOf attribute to Treatment base class
10. To test the examples for the treatment types I am using rdf:Description and rdf:Type similar to what we had in previous examples.
11. Moving on we have two properties ProvidedBy which relates Treatments to Providers and RecievedBy for Patients
12. We can test this using props ProvidedBy and props RecievedBy
13. For the RadiologistProvided since it is a form of Treatment we can use elems RadiologistProvided to get all the instances of that class
14. Same for the RadiologyPatient
15. Finally for each patient to have a unique id I have used FunctionalProperty with domain and type which points to InverseFunctionalProperty, it should make a constraint on patientId attribute.
16. I have validate the OWL using the provided owl.jar
17. All the triples are found in **rdf-triples** file and queries in **rdf-queries**