A hierarchical organization of classes has a very important practical significance, which we outline now. Consider the range restriction

People can only rent residential units.

Suppose Baron Way Apartment is defined as an apartment. Then, according to the preceding restriction, it does not qualify as a Residential Unit because there is no statement specifying that the Baron Way Apartment is also a residential unit. It would be counterintuitive to overcome this difficulty by adding that statement to our description. Instead we would like the Baron Way Apartment to *inherit* the ability to be rented from the class of residential units. Exactly this is done in RDF Schema.

By doing so, RDF Schema *fixes the semantics* of "is a subclass of." Now it is not up to an application to interpret "is a subclass of;" instead its intended meaning must be used by all RDF processing software. By making such semantic definitions, RDFS is a (still limited) language for defining the semantics of particular domains. Stated another way, RDF Schema is a primitive *ontology language*.

Classes, inheritance, and properties are, of course, known in other fields of computing – for example, in object-oriented programming. But while there are many similarities, there are differences, too. In object-oriented programming, an object class defines the properties that apply to it. To add new properties to a class means to modify the class.

However, in RDFS, properties are defined globally. That is, they are not encapsulated as attributes in class definitions. It is possible to define new properties that apply to an existing class without changing that class.

On one hand, this is a powerful mechanism with far-reaching consequences: we may use classes defined by others and adapt them to our requirements through new properties. On the other hand, this handling of properties deviates from the standard approach that has emerged in the area of modeling and object-oriented programming. It is another idiosyncratic feature of RDF/RDFS.