CECS 277 – Lecture 8 – Lists, Sets, and Maps

Lists, Sets, and Maps are all types of collections. Collections hold groups of objects, known as elements, and are usually expand and contract as new elements are added and removed. The differences between each of these are that the collection types store the objects in different ways, and have methods of accessing, storing, and removing the elements. Which one you choose depends on what the needs for the program are.

List – A list is a type of collection that is used when sequence or index position matters. If you are storing a list of items in a particular order, use a type of list. A few types of lists are: ArrayList, LinkedList, Stack, and Vector.

Example: ArrayList

```
ArrayList<Animal> animals = new ArrayList<Animal>();
animals.add(new Cat("Fluffy"));
animals.add(new Dog("Spot"));
Animal a = animals.get(0);
animals.clear();
```

Set – A set is a type of collection that is used when uniqueness matters, as they do not allow any duplicate values. A few types of sets are: HashSet, and TreeSet:

HashSet – In order to use a HashSet, the elements of your set must be hashable (your class must provide a hashCode() method). A hashable value is one that can be used to calculate a new value that will then be used as a storage location in the set. HashSets are usually known for their ability to store values very quickly. Several STL types already have the hashCode() method included: String, Integer, Point, Rectangle, Color, and all of the collection classes.

Example: A set of strings stored as a HashSet.

```
Set<String> food = new HashSet<String>();
food.add("Pizza");
food.add("Ice Cream");
for( String s : food )
     System.out.println(s);
```

TreeSet – A TreeSet is used when you want your elements to be stored in sorted order. Because of this, TreeSet insertions can be costly and time consuming, so should be avoided except when necessary. The class of objects being stored into the set need to implement the Comparable interface and have a compareTo() method (the String and Integer classes have this implemented).

Example: A set of strings stored as a TreeSet.

```
Set<String> food = new TreeSet<String>();
food.add("Pizza");
food.add("Ice Cream");
System.out.println(food.first());
System.out.println(food.last());
```

Map – A map is a type of collection that stores elements based on a key-value pair. A key-value pair is one that associates a key to a value in a many-to-one fashion. There may not be any duplicate keys in a map, but several keys may map to the same value. A few types of maps are: HashMap, and TreeMap:

HashMap – In order to use a HashMap, the key elements of your map must be hashable (your class must provide a hashCode() method). Several STL types already have the hashCode() method included: String, Integer, Point, Rectangle, Color, and all of the collection classes.

Example: A set of strings stored as a HashMap.

```
Map <String, Integer> grades =
                new HashMap <String, Integer>();
     grades.put("Stewart", 92);
     grades.put("Mary", 80);
     grades.put("George", 67);
     grades.put("Scott", 80);
     int s = grades.size();
     if(grades.containsValue(100)){
          System.out.println("Perfect score");
```

TreeMap – A TreeMap is used when you want your elements to be stored in sorted order. Because of this, TreeMap insertions can be costly and time consuming, so should be avoided except when necessary. The class of objects being stored in the map as the key need to implement the Comparable interface and have a compareTo() method (the String and Integer classes have this implemented).

Example: A set of strings stored as a TreeMap.

```
Map<String, Integer> grades =
          new TreeMap<String, Integer>();
grades.put("Stewart", 92);
grades.put("Mary", 80);
grades.put("George", 67);
grades.put("Scott", 80);
Set <String> keys = grades.keySet();
for( String s : keys ){
     Integer i = grades.get(s);
     System.out.println(s+ " got a "+i);
}
```