CSE 341 Section 10 Solutions

1. Sketch the class definition and method signatures for a Dictionary class, which allows one to store or look up a value indexed by a key. Give the method signatures for get, put, isEmpty, keys, and values. The last two methods should return parameterized collections. (This class is similar to the builtin class HashMap in the Java collections library.)

Answer:

```
class Dictionary<K,V>
V get(K key)
void put(K key, V value)
boolean isEmpty()
Collection<K> keys()
Collection<V> values()
```

2. Joe Mocha is defining an interface Appendable that includes an append method. He then defines two classes, MyString and MyList, which both implement Appendable. He wants Java's type system to allow a MyString to be appended to a MyString, and a MyList to be appended to a MyList, but not a MyString to a MyList, or a MyList to a MyString.

Here is his definition of Appendable:

```
interface Appendable {
   Appendable append(Appendable a);
}
```

What is wrong with this definition? What is a correct one? Also write a definition for a class MyString that uses the revised definition of Appendable. (Just put . . . in the body of the method — we only care about the header.)

Answer:
The problem is that Appendable doesn't have any information about the type of the items in the collections being appended. The solution is to use generics.

```
interface Appendable<A> {
   Appendable<A> append(Appendable<A> a)
}

class MyString implements Appendable<MyString> {
   public Appendable<MyString> append(Appendable<MyString> a) {
      ....
   }
}
```

3. Consult the “Wild4” class on the back of this page. Suppose we have implemented Java for classes Point and ColorPoint, just like the ruby classes from a previous exercise (but all you need to know is that ColorPoint extends class Point).

a. Which of the calls in the main method work in Java?
b. Suppose Java generics supported covariance, but not contravariance. Which of the calls in the main method would work then?
c. Suppose Java generics supported contravariance, but not covariance. Which of the calls in the main method would work then?
d. Write method signatures (you don’t have to write the body) for methods print3 and print4. print3 should accept any type of LinkedList, and print4 should only accept LinkedList<Point> and LinkedList<ColorPoint>.
e. Discussion Question: When might you want to accept parameters like in print3? print4? Can you think of example use-cases that would make these sorts of methods useful?

Answer:

a. 1 and 4
b. 1, 3, and 4 (covariance allows you to supply a more specific type than is specified)
c. 1, 2, and 4 (contravariance allows you to supply a less specific type than is specified)

d.
```
public static void print3(LinkedList<?> lst) { }

public static void print4(LinkedList<? extends Point> lst) { }
```

e. print3 is useful if you assume your method is useful for all **Objects.** Methods like this include printing (all objects have a toString()), or storing in a HashTable (all objects have a hashcode()).

   print4 and is useful if you want to be able to assume the argument contains Points or ColorPoints. Methods that’d come in handy would be ones that run some computations on the x/y fields.

```
// CSE 341
// Example of generics and contravariance/covariance

class Wild4 {

   public static void printAll(LinkedList<Point> s) {
         for (Point e : s) {
            System.out.println(e);
         }
   }

   public static void printAll2(LinkedList<ColorPoint> s) {
         for (Point e : s) {
            System.out.println(e);
         }
   }

   public static void main(String[] args) {
         LinkedList<Point> plist = new LinkedList<Point>();
         LinkedList<ColorPoint> clist = new LinkedList<ColorPoint>();
         plist.add(new Point(10,20));
         clist.add(new ColorPoint(30,50,"purple"));
         printAll(plist);   // 1
         printAll2(plist); // 2
         printAll(clist);   // 3
         printAll2(clist); // 4
   }
}
```