1. Consider the following Java code fragments. (The first 3 lines are the same for all of them; it’s just the last line that is different.) In each case, does the code compile correctly? If so, does it execute without error, or is there an exception?

```java
Point[] a = new Point[10];
Object[] b;
b = a;
b[0] = new Point(10,20);
```

- Compiles and executes without error.

```java
Point[] a = new Point[10];
Object[] b;
b = a;
b[0] = "hi there";
```

- Compiles but gets a run-time ArrayStoreException.

```java
Point[] a = new Point[10];
Object[] b;
b = a;
a[0] = "hi there";
```

- Gets a compile-time error.

2. What about code that is analogous to that in Question 1, but that uses `ArrayList`? For example:

```java
ArrayList<Point> a = new ArrayList<Point>();
ArrayList<Object> b;
b = a;
b.add(new Point(10,20));
```

These all get a compile-time error complaining about the `b = a` assignment.

3. Sketch the class definition and method signatures for a Stack class, parameterized by the type of element on the stack. Give the method signatures for `push`, `pop`, and `isEmpty`.

```java
class Stack<E>

void push(E element)
E pop()
boolean isEmpty()
```

4. 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.)
class Dictionary<K,V>

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

5. 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?

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<E> {
  Appendable<E> append(Appendable<E> a)
}

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.)

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