§1 Inheritance (not official names I think)

Definition — Inheritance is the parent-child stuff we learned in class, essentially how a class can *inherit* stuff from its parent classes.

There are two different types of inheritance that we learned: *implements* and *extends*.

1. Implements is where the parent class is initialized without a constructor, and instead with empty methods in an interface class. When a child-class *implements* an interface, it has to use the methods defined in the parent class.

```java
      public interface Name {
        void methodName();
      }
```

```java
      public class hi implements Name {
        public void methodName() {
          //stuff
        }
      }
```

   A class can implement > 1 interface, but it will be required to have all the classes the interfaces had. Some important notes for the AP Exam:

   - Will not compile unless format follows above
   - Interface methods are abstract, which means it's guranteeing that all subclasses need to have and run this method (or it won't compile).

2. Extends is where the parent class is a normal class, and a subclass will extend it, which essentially means that the subclass will have, and can use, all the classes that it has *extended* from the parent class.

```java
        public class Name {
          public Name() {
          }
          public void hi() {
          }
        }
```

```java
        public class hi extends Name {
          public Name2() {
          }
        }
```

   Running the 'super()' method in the child class calls upon the parent class. If the parent method is 'private', then that means child classes cannot access it; however, if its 'public' or 'protected', they can. Something very important is if the child class **doesn't explicitly invoke superclass constructor**, it'll insert an *implicit* call to the parent class's constructor.
   Another important thing is how a child class can only extend one other class. This leads us to our most tricky part of inheritance: polymorphism.

   Definition — Polymorphism: How repeated methods behave in child-parent class inheritance stuff (i think)

   In a child class, if it extends a parent class, and a parent class has the same method and the method is called, it'll always call upon the child classes's thing (overriding the parents's method). However, for this to compile, it must have the exact same methodname, returntype, . . . as the super class.

```java
public class Name {
  private int a;
  public Name(int a) {
    this.a = a;
  }
  public void hello() {
    System.out.println(a);
  }
}
```

```java
public class hi extends Name {
  private int hivar;
  public hi(int a, int hivar) {
    super(a);
    this.hivar = hivar;
  }
  public void hello() {
    System.out.println(hivar);
  }
}
```

```java
public static void main(String[] args) {
  Name a = new hi(3, 4);
  a.hello(); //should output 4
}
```

This brings us to initialization. In the previous example, a 'Name' datatype object was initialized, but even though its a 'hi' object, it wouldn't have been able to call 'hello()' unless the 'Name' had a method called 'hello()' (parents can't access child classes but the opposite is true, but this can be worked around by casting).
Other important stuff abt this:

- In arrays, you can add child class to parent class initialized type arrays, but you can't do the opposite (and when you add a child class, it kinda casts it to the parent type and you can't use the child-specific methods I think.
- When a child class calls a method in the parent class that calls a method in the parent class that's been overridden by the child class, it'll go to the child class instead even though the method was called in the parent class as the original object is the child class type.

§2 Other Stuff

Everything is good except paying close attention to the problem, as well as for loop heads, and making sure that **removing elements from arraylist shifts the index**.