An Appetizer

Consider the following code:

```
public abstract class Food implements Consumable {
	String name;
	public void play() {
		System.out.println("Mom said don't play with your food.");
	}
}

public interface Consumable {
	public void prepare();
	public void eat();
	public void discard();
}

class Snack extends Food {
	public Snack(String name) {
		this.name = name;
	}
	public void prepare() {
		System.out.println("Taking " + this.name + " out of wrapper");
	}
	public void eat() {
		System.out.println("Snacking on " + this.name);
	}
	public void discard() {
		System.out.println("Throwing out " + this.name);
	}
	public void play() {
		super.play();
	}
}
```

```
class Meal extends Food {
	public Meal(String name) {
		this.name = name;
	}
	public void prepare() {
		System.out.println("Cooking " + this.name);
	}
	public void eat() {
		System.out.println("Eating " + this.name);
	}
	public void discard() {
		System.out.println("Cleaning utensils from " + this.name);
	}
	public void play() {
		super.play();
	}
}
```

Questions

1. What is the difference between an interface and an abstract class?
    a. Interfaces guarantees that a set of functions are used between two classes, whereas abstract classes are more for inheritance purposes from an object that itself should be created.
    b. Multiple Inheritance is not allowed in Java, but classes can have multiple interfaces.
    c. Abstract classes can have some implementation code (like the play function in Food), but interfaces cannot.
2. Do we need the play() method in Snack and Meal?
    a. No, we do not need the play method because the abstract class already defined it, and Java will just call the child class's parent class method if it cannot find it in the child class.
    b. This is an example of why you would want to write abstract method's implementation code and why inheritance is important, because for Snack and Meal you'd have to rewrite the same method in both of them instead of just in the parent class.
3. What is another class that would utilize the Consumable interface and inherits from Food?
    a. Dessert/Pastry
    b. Breakfast/Lunch/Dinner can inherit from Meal which inherits food
    c. many more possibilities
4. Is there a class that would inherit from Food, but not utilize Consumables?
    a. Trick question. All subclasses of Food have to be Consumable. The compiler requires you to implement the methods of Consumable in any non-abstract class that extends Food!
    b. You do not need to write "implements" for Snack and Meal for example.

The Iterator Interface

In Java, an iterator is an object which allows us to traverse once through a data structure in a linear fashion. Each iterator has two methods, `hasNext`, and `next`. (Notice the Iterator interface is different in Java vs Python)

```
interface Iterator {
  boolean hasNext(); # returns whether or not there are items left
  Object next();     # returns the next item
                     # undefined behavior if hasNext() is false
}
```

Say we wanted to make iterators for arrays and `SLists` (It may not be clear why we would want to do this, but in the future, this will come in handy for iterating through non-linear data structures, like Trees and Graphs.) Focusing on arrays for now, we want the following behavior to work:

```
int[] arr = new int[]{1, 2, 3, 4, 5, 6};
IntArrayIterator iter = new IntArrayIterator(arr);

System.out.println(iter.hasNext());        // Outputs True
System.out.println(iter.next());           // Outputs 1

System.out.println((int) iter.next() + 3); // Outputs 5
# Why did we have to cast here?
Since iter.next() has static type Object by default, we must cast it
to an int in order to add it to 3.

// Outputs 3, 4, 5, 6, each on a different line
while (iter.hasNext()) {
    System.out.println(iter.next());
}
```

Fill in the following so that the above is achieved. Hint: what does the iterator need to remember in order for `hasNext` and `next` to work?

There are multiple ways to solve this. I've included three different implementations, with notes. There are many more variations. -- Andrew

```
class IntArrayIterator implements Iterator {
    private int index;
    private int[] array;
    public IntArrayIterator(int[] arr) {
        array = arr;
        index = 0;
    }

    public boolean hasNext() {
        return index < array.length;
    }

    public Object next() {
        Object val = array[index]; // Save the return value
        index += 1;                // Increment index so that next
        return val                 // call to next() will work
    }
}
```

The above code is completely reasonable, but next() was a little bit verbose. Here's one way to avoid that.

```
class IntArrayIterator2 implements Iterator {
    private int index;
    private int[] array;
    public IntArrayIterator2(int[] arr) {
        array = arr;
        index = 0;
    }

    public boolean hasNext() {
        return index < array.length - 1;
    }

    public Object next() {
        index += 1;                // Increment first
        return array[index -1]     // avoid having to use val
    }
}
```

```
class IntArrayIterator3 implements Iterator {
    private int index;
    private int array;
    public IntArrayIterator3(int[] arr) {
        array = arr;
        index = 0;
    }

    public boolean hasNext() {
        return index < array.length - 1;
    }

    public Object next() {
        return array[index++]        // Java trickery*
    }
}
```

*http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
Don't bother with this solution unless you have extra time to read the link. Relevant info is on the very bottom of that page.

Let's do the same for `IntLists`. This time, no starter code.

```
class IntListIterator implements Iterator {
    private IntList ptr;
    public IntListIterator(IntList l) {
        ptr = l;
    }
    public boolean hasNext() {
        return ptr != null;
    }
    public Object next() {
        Object val = ptr.head;
        ptr = ptr.tail;
        return val;
    }
}
```

Now say we wanted to write *one* method that printed out every element of an iterator, regardless of it was an `SListIterator` or an `IntArrayIterator`. How would we do that? Write `printAll`.

```
public static void printAll(Iterator i) {
    while (i.hasNext()) {
        System.out.println(i.next());
    }
}
```