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?

2. Do we need the play() method in Snack and Meal?

3. What is another class that would utilize the Consumable interface and inherits from Food?

4. Is there a class that would inherit from Food, but not utilize Consumables?

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 `IntLists` (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?

// 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?

```
class IntArrayIterator implements Iterator {



   public ArrayIterator(int[] arr) {




   }

   public boolean hasNext() {



   }

   public Object next() {




   }
}
```

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

```
class IntListsIterator implements Iterator {









}
```

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