Framework for the Account class

Make no assumptions about methods not mentioned in the outline below.

```java
public class Account {
    // This class represents a bank account whose current balance
    // is a nonnegative amount in US dollars, which may have
    // an auxiliary account to provide overdraft protection.
    // Initialize an Account object with the given balance.
    // Precondition: balance > 0.
    public Account (int balance) {
        myBalance = balance;
        myParent = null;
    }
    // Initialize an Account object with the given balance,
    // and the given auxiliary account for overdraft protection.
    // Precondition: balance > 0.
    public Account (int balance, Account overdraftAcct) {
        myBalance = balance;
        myParent = overdraftAcct;
    }
    // Add the given amount to this account's balance.
    // Precondition: amount > 0.
    public void deposit (int amount) {
        myBalance += amount;
    }
    // If this account's balance is at least the given amount,
    // deduct the amount from the balance and return true.
    // If there is no overdraft protection, print an error message and return false.
    // Otherwise, request a withdrawal of the given amount from the parent account,
    // and return true if that withdrawal succeeds and false otherwise.
    // Precondition: amount ≥ 0.
    public boolean withdraw (int amount) {
        ...
    }
    // Returns the number of dollars in this account.
    public int balance ( ) {
        return myBalance;
    }
    private int myBalance;
    private Account myParent;
}
```
Framework for the Interval class

Make no assumptions about methods not mentioned in the outline below.

public class Interval {
    private int myLeft;
    private int myRight;

    public Interval (int left, int right) throws IllegalArgumentException {
        if (left > right) {
            throw new IllegalArgumentException ("Illegal empty interval: [" + left + "," + right + "]");
        }
        myLeft = left;
        myRight = right;
    }

    // Methods to enumerate the integers in an interval.
    // Variables to keep track of the enumeration would be declared here.
    public void initEnumeration ( ) {
        ...
    }
    public boolean hasMoreElements ( ) {
        ...
    }
    public int nextElement ( ) {
        ...
    }

    // Return true when this interval and the argument interval overlap,
    // that is, share at least one integer, or are adjacent, that is, the largest integer
    // in one of the intervals is 1 less than the smallest integer in the other.
    // Return false otherwise.
    public boolean combinable (Interval intvl) {
        ...
    }

    // Precondition: The argument interval can be combined with this interval.
    // Extend this interval so that it contains all the integers in the argument,
    // plus all the integers it contained previously.
    public void combine (Interval intvl) {
        ...
    }

    // Return the string representation of an interval.
    public String toString ( ) {
        return "][" + myLeft + "," + myRight + "]";
    }

    // Return true if this interval is equal to the argument; return false otherwise.
    public boolean equals (Object obj) {
        ...
    }
}
**Framework for the** IntervalSequence **class**

Make no assumptions about methods not mentioned in the outline below.

```java
public class IntervalSequence {
    private Interval[] myIntervals;
    private int myCount;
    // Initialize a sequence to hold at most the given number of intervals.
    public IntervalSequence (int capacity) {
        ...
    }
    // Return true if this sequence is empty; return false otherwise.
    public boolean isEmpty ( ) {
        ...
    }
    // Return the number of intervals in this sequence.
    public int size ( ) {
        ...
    }
    // Return the interval at the given position in this sequence.
    // Precondition: 0 ≤ pos ≤ size()-1.
    public Interval elementAt (int pos) {
        ...
    }
    // Add the argument interval to the end of this sequence.
    // Precondition: size() < the capacity of the sequence.
    public void add (Interval toAdd) {
        ...
    }
    // Remove from this sequence the interval at the given position.
    // Precondition: 0 ≤ pos ≤ size()-1.
    public void remove (int pos) {
        ...
    }
    // Insert into this sequence the given interval at the given position.
    // Precondition: 0 ≤ pos ≤ size() and size() < the capacity of this sequence.
    public void insert (Interval toInsert, int pos) {
        ...
    }
}
```