Hash Tables

Separate Chaining from Textbook

```
public class Employee
{
   public boolean equals(Object rhs)
    { return rhs instanceof Employee &&
        name.equals(  ((Employee)rhs).name ); }
   public int hashCode()
    { return name.hashcode(); }

   private String name;
   private double salary;
   private int seniority;
}
```

```
public class SeparateChainingHashTable<AnyType>
{
    private static final int DEFAULT_TABLE_SIZE = 101;
    private List<AnyType> [ ] theLists;
    private int currentSize;
    public SeparateChainingHashTable( ){}
    public SeparateChainingHashTable( int size ){}
    public void insert( AnyType x ){}
    public void remove( AnyType x ){}
    public boolean contains( AnyType x ){}
}
```

```
public SeparateChainingHashTable( )
{
    this( DEFAULT_TABLE_SIZE );
}

public SeparateChainingHashTable( int size )
{
    theLists = new LinkedList[ nextPrime( size ) ];
    for( int i = 0; i < theLists.length; i++ )
        theLists[ i ] = new LinkedList<AnyType>( );
}
```

Table Location of an Item

```
private int myhash( AnyType x )
{
    int hashVal = x.hashCode( );

    hashVal %= theLists.length;
    if( hashVal < 0 )
        hashVal += theLists.length;

    return hashVal;
}
```

```
public void insert( AnyType x )
{
    List<AnyType> whichList = theLists[ myhash( x ) ];
    if( !whichList.contains( x ) )
    {
        whichList.add( x );

        Code for increasing table size if necessary
    }
}
```

```
public boolean contains( AnyType x )
{
    List<AnyType> whichList = theLists[ myhash( x ) ];
    return whichList.contains( x );
}
```

```
public void remove( AnyType x )
{
    List<AnyType> whichList = theLists[ myhash( x ) ];
    if( whichList.contains( x ) )
    {
        whichList.remove( x );
        currentSize--;
    }
}
```

Using the Hash Table

```
public static void main( String [ ] args )
{
    SeparateChainingHashTable<Integer> H = new
                        SeparateChainingHashTable<Integer>( );
    final int NUMS = 40000;
    final int GAP  =   37;

    for( int i = GAP; i != 0; i = ( i + GAP ) % NUMS )
        H.insert( i );
    for( int i = 1; i < NUMS; i+= 2 )
        H.remove( i );
    for( int i = 2; i < NUMS; i+=2 )
        if( !H.contains( i ) )
            System.out.println( "Find fails " + i );

    for( int i = 1; i < NUMS; i+=2 )
    {
        if( H.contains( i ) )
            System.out.println( "OOPS!!! " +  i  );
    }
}
```