```
// The Itr class is a private Inner Class found in the AbstractList class

private class Itr implements Iterator<E> {
    /**
     * Index of element to be returned by subsequent call to next.
     */
    int cursor = 0;

    /**
     * Index of element returned by most recent call to next or
     * previous.  Reset to -1 if this element is deleted by a call
     * to remove.
     */
    int lastRet = -1;

    /**
     * The modCount value that the iterator believes that the backing
     * List should have.  If this expectation is violated, the iterator
     * has detected concurrent modification.
     */
    int expectedModCount = modCount;

    public boolean hasNext()
    {
       return cursor != size();
    }

    public E next()
    {
       checkForComodification();
       try
       {
          E next = get(cursor);
          lastRet = cursor++;
          return next;
        }
        catch (IndexOutOfBoundsException e)
        {
          checkForComodification();
          throw new NoSuchElementException();
        }
    }

    public void remove()
    {
       if (lastRet == -1)
          throw new IllegalStateException();
       checkForComodification();

       try
       {
          AbstractList.this.remove(lastRet);
          if (lastRet < cursor)
             cursor--;
          lastRet = -1;
          expectedModCount = modCount;
       }
```

```
      catch (IndexOutOfBoundsException e)
      {
         throw new ConcurrentModificationException();
      }
   }

   final void checkForComodification()
   {
      if (modCount != expectedModCount)
      throw new ConcurrentModificationException();
   }
    }

// The ListItr class is a subclass of the Itr class and is also a private
// Inner Class found in the AbstractList class

private class ListItr extends Itr implements ListIterator<E>
{
   ListItr(int index)
   {
      cursor = index;
   }

   public boolean hasPrevious()
   {
      return cursor != 0;
   }

   public E previous()
   {
      checkForComodification();
      try
      {
         int i = cursor - 1;
         E previous = get(i);
         lastRet = cursor = i;
         return previous;
      }
      catch (IndexOutOfBoundsException e)
      {
         checkForComodification();
         throw new NoSuchElementException();
      }
   }

   public int nextIndex()
   {
      return cursor;
   }

   public int previousIndex()
   {
      return cursor-1;
   }

   public void set(E e)
   {
```

```
      if (lastRet == -1)
         throw new IllegalStateException();
      checkForComodification();

      try
      {
         AbstractList.this.set(lastRet, e);
         expectedModCount = modCount;
      }
      catch (IndexOutOfBoundsException ex)
      {
         throw new ConcurrentModificationException();
      }
   }

   public void add(E e)
   {
      checkForComodification();

      try
      {
         AbstractList.this.add(cursor++, e);
         lastRet = -1;
         expectedModCount = modCount;
      }
      catch (IndexOutOfBoundsException ex)
      {
         throw new ConcurrentModificationException();
      }
   }
}
```