```java
import java.util.Iterator;

/**
 * {@code Stack} represented as a singly linked list, done "bare-handed", with
 * implementations of primary methods.
 *
 * <p>
 * Execution-time performance of all methods implemented in this class is O(1).
 *
 * @param <T>
 *            type of Stack entries
 * @convention <pre>
 * $this.length >= 0  and
 * if $this.length = 0 then
 *   [$this.top is null]
 * else
 *   [$this.top is not null]  and
 *   [$this.top points to the first node of a singly linked list
 *    containing $this.length nodes]  and
 *   [next in the last node of that list is null]
 * </pre>
 * @correspondence this = [data in $this.length nodes starting at $this.top]
 */
public class Stack2<T> extends StackSecondary<T> {

    /*
     * Private members --------------------------------------------------------
     */

    /**
     * Node class for singly linked list nodes.
     */
    private final class Node {

        /**
         * Data in node.
         */
        private T data;

        /**
         * Next node in singly linked list, or null.
         */
        private Node next;

    }

    /**
     * Top node of singly linked list.
     */
    private Node top;

    /**
     * Number of nodes in singly linked list, i.e., length = |this|.
     */
    private int length;

    /**
```

```
     * Checks that the part of the convention repeated below holds for the
     * current representation.
     *
     * @return true if the convention holds (or if assertion checking is off);
     *         otherwise reports a violated assertion
     * @convention <pre>
     * $this.length >= 0  and
     * if $this.length == 0 then
     *   [$this.top is null]
     * else
     *   [$this.top is not null]  and
     *   [$this.top points to the first node of a singly linked list
     *    containing $this.length nodes]  and
     *   [next in the last node of that list is null]
     * </pre>
     */
    private boolean conventionHolds() {
        assert this.length >= 0 : "Violation of: $this.length >= 0";
        if (this.length == 0) {
            assert this.top == null : ""
                    + "Violation of: if $this.length == 0 then $this.top is null";
        } else {
            assert this.top != null : ""
                    + "Violation of: if $this.length > 0 then $this.top is not null";
            int count = 0;
            Node tmp = this.top;
            while ((tmp != null) && (count < this.length)) {
                count++;
                tmp = tmp.next;
            }
            assert this.length == count : "Violation of: if $this.length > 0 then "
                    + "[$this.top points to the first node of a singly "
                    + "linked list containing $this.length nodes]";
            assert tmp == null : "Violation of: if $this.length > 0 then "
                    + "[$this.top points to the first node of a singly "
                    + "linked list containing $this.length nodes] and "
                    + "[next in the last node of that list is null]";
        }
        return true;
    }

    /**
     * Creator of initial representation.
     */
    private void createNewRep() {

        this.top = new Node();
        this.length = 1;

    }

    /*
     * Constructors -----------------------------------------------------------
     */

    /**
     * No-argument constructor.
```

```java
     */
    public Stack2() {
        this.createNewRep();
        assert this.conventionHolds();
    }

    /*
     * Standard methods -------------------------------------------------------
     */

    @SuppressWarnings("unchecked")
    @Override
    public final Stack<T> newInstance() {
        try {
            return this.getClass().getConstructor().newInstance();
        } catch (ReflectiveOperationException e) {
            throw new AssertionError(
                    "Cannot construct object of type " + this.getClass());
        }
    }

    @Override
    public final void clear() {
        this.createNewRep();
        assert this.conventionHolds();
    }

    @Override
    public final void transferFrom(Stack<T> source) {
        assert source != null : "Violation of: source is not null";
        assert source != this : "Violation of: source is not this";
        assert source instanceof Stack2<?> : ""
                + "Violation of: source is of dynamic type Stack2<?>";
        /*
         * This cast cannot fail since the assert above would have stopped
         * execution in that case: source must be of dynamic type Stack2<?>, and
         * the ? must be T or the call would not have compiled.
         */
        Stack2<T> localSource = (Stack2<T>) source;
        this.top = localSource.top;
        this.length = localSource.length;
        localSource.createNewRep();
        assert this.conventionHolds();
        assert localSource.conventionHolds();
    }

    /*
     * Kernel methods ---------------------------------------------------------
     */

    @Override
    public final void push(T x) {
        assert x != null : "Violation of: x is not null";

        // create new node
        Node temp = new Node();
        temp.data = x;

        // rearrange nodes
        temp.next = this.top;
        this.top = temp;

        this.length++;

        assert this.conventionHolds();
    }

    @Override
    public final T pop() {
        assert this.length() > 0 : "Violation of: this /= <>";

        T temp = this.top.data;

        this.top = this.top.next;

        this.length--;

        assert this.conventionHolds();
        // Fix this line to return the result after checking the convention.
        return temp;
    }

    @Override
    public final int length() {

        assert this.conventionHolds();
        // Fix this line to return the result after checking the convention.
        return this.length;
    }

    @Override
    public final Iterator<T> iterator() {
        return new Stack2Iterator();
    }

    /**
     * Implementation of {@code Iterator} interface for {@code Stack2}.
     */
    private final class Stack2Iterator implements Iterator<T> {

        /**
         * Current node in the linked list.
         */
        private Node current;

        /**
         * No-argument constructor.
         */
        private Stack2Iterator() {
            this.current = Stack2.this.top;
            assert Stack2.this.conventionHolds();
        }

        @Override
```

```java
        public boolean hasNext() {
            assert Stack2.this.conventionHolds();
            return this.current != null;
        }

        @Override
        public T next() {
            assert this.hasNext() : "Violation of: ~this.unseen /= <>";
            if (!this.hasNext()) {
                /*
                 * Exception is supposed to be thrown in this case, but with
                 * assertion-checking enabled it cannot happen because of assert
                 * above.
                 */
                throw new NoSuchElementException();
            }
            T x = this.current.data;
            this.current = this.current.next;
            assert Stack2.this.conventionHolds();
            return x;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException(
                    "remove operation not supported");
        }

    }

}

```