```java
import java.lang.reflect.Constructor;

/**
 * {@code SortingMachine} represented as a {@code Queue} (using an embedding of
 * insertion sort), with implementations of primary methods.
 *
 * @param <T>
 *            type of {@code SortingMachine} entries
 * @mathdefinitions <pre>
 * IS_TOTAL_PREORDER (
 *   r: binary relation on T
 *  ) : boolean is
 *  for all x, y, z: T
 *   ((r(x, y) or r(y, x))  and
 *    (if (r(x, y) and r(y, z)) then r(x, z)))
 *
 * IS_SORTED (
 *   s: string of T,
 *   r: binary relation on T
 *  ) : boolean is
 *  for all x, y: T where (<x, y> is substring of s) (r(x, y))
 * </pre>
 * @convention <pre>
 * IS_TOTAL_PREORDER([relation computed by $this.machineOrder.compare method])  and
 * IS_SORTED($this.entries, [relation computed by $this.machineOrder.compare method])
 * </pre>
 * @correspondence <pre>
 * this =
 *   ($this.insertionMode, $this.machineOrder, multiset_entries($this.entries))
 * </pre>
 */
public class SortingMachine3<T> extends SortingMachineSecondary<T> {

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

    /**
     * Insertion mode.
     */
    private boolean insertionMode;

    /**
     * Order.
     */
    private Comparator<T> machineOrder;

    /**
     * Entries.
     */
    private Queue<T> entries;

    /**
     * Creator of initial representation.
     *
     * @param order
     *            total preorder for sorting
     */
    private void createNewRep(Comparator<T> order) {
        this.insertionMode = true;
        this.machineOrder = order;
        this.entries = new Queue1L<T>();
    }

    /**
     * Inserts the given {@code T} in the {@code Queue<T>} sorted according to
     * the given {@code Comparator<T>} and maintains the {@code Queue<T>}
     * sorted.
     *
     * @param <T>
     *            type of {@code Queue} entries
     * @param q
     *            the {@code Queue} to insert into
     * @param x
     *            the {@code T} to insert
     * @param order
     *            the {@code Comparator} defining the order for {@code T}
     * @updates q
     * @requires <pre>
     * IS_TOTAL_PREORDER([relation computed by order.compare method])  and
     * IS_SORTED(q, [relation computed by order.compare method])
     * </pre>
     * @ensures <pre>
     * perms(q, #q * <x>)  and
     * IS_SORTED(q, [relation computed by order.compare method])
     * </pre>
     */
    private static <T> void insertInOrder(Queue<T> q, T x,
            Comparator<T> order) {
        assert q != null : "Violation of: q is not null";
        assert x != null : "Violation of: x is not null";
        assert order != null : "Violation of: order is not null";

        boolean in = false;
        int idx = 0;

        // while index less than length and x not inserted into queue
        while (idx < q.length() && !in) {
            // if the front value is alphabetically less than or equal to x
            if (order.compare(q.front(), x) <= 0) {
                q.enqueue(x);
                in = true;
            }

            idx++;
            // move front to back
            T temp = q.dequeue();
            q.enqueue(temp);
        }

        // put queue back in order
        while (idx < q.length()) {
            T temp = q.dequeue();
            q.enqueue(temp);
            idx++;
        }

    }

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

    /**
     * Constructor from order.
     *
     * @param order
     *            total preorder for sorting
     */
    public SortingMachine3(Comparator<T> order) {
        this.createNewRep(order);
    }

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

    @SuppressWarnings("unchecked")
    @Override
    public final SortingMachine<T> newInstance() {
        try {
            Constructor<?> c = this.getClass().getConstructor(Comparator.class);
            return (SortingMachine<T>) c.newInstance(this.machineOrder);
        } catch (ReflectiveOperationException e) {
            throw new AssertionError(
                    "Cannot construct object of type " + this.getClass());
        }
    }

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

    @Override
    public final void transferFrom(SortingMachine<T> source) {
        assert source != null : "Violation of: source is not null";
        assert source != this : "Violation of: source is not this";
        assert source instanceof SortingMachine3<?> : ""
                + "Violation of: source is of dynamic type SortingMachine3<?>";
        /*
         * This cast cannot fail since the assert above would have stopped
         * execution in that case: source must be of dynamic type
         * SortingMachine3<?>, and the ? must be T or the call would not have
         * compiled.
         */
        SortingMachine3<T> localSource = (SortingMachine3<T>) source;
        this.insertionMode = localSource.insertionMode;
        this.machineOrder = localSource.machineOrder;
        this.entries = localSource.entries;
        localSource.createNewRep(localSource.machineOrder);
    }

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

    @Override
    public final void add(T x) {
        assert x != null : "Violation of: x is not null";
        assert this.isInInsertionMode() : "Violation of: this.insertion_mode";

        this.insertInOrder(this.entries, x, this.machineOrder);

    }

    @Override
    public final void changeToExtractionMode() {
        assert this.isInInsertionMode() : "Violation of: this.insertion_mode";

    }

    @Override
    public final T removeFirst() {
        assert !this
                .isInInsertionMode() : "Violation of: not this.insertion_mode";
        assert this.size() > 0 : "Violation of: this.contents /= {}";

        // TODO #4 - remove and return first entry in machine contents

        // This line added just to make the component compilable.
        return null;
    }

    @Override
    public final boolean isInInsertionMode() {

        // TODO #5 - report whether machine is in insertion mode

        // This line added just to make the component compilable.
        return false;
    }

    @Override
    public final Comparator<T> order() {

        // TODO #6 - report order used by machine

        // This line added just to make the component compilable.
        return null;
    }

    @Override
    public final int size() {

        return this.entries.length();

    }

    @Override
    public final Iterator<T> iterator() {
        return this.entries.iterator();
    }

}

```