```java
import components.naturalnumber.NaturalNumber;
import components.naturalnumber.NaturalNumberSecondary;

/**
 * {@code NaturalNumber} represented as a {@code String} with implementations of
 * primary methods.
 *
 * @convention <pre>
 * [all characters of $this.rep are '0' through '9']  and
 * [$this.rep does not start with '0']
 * </pre>
 * @correspondence <pre>
 * this = [if $this.rep = "" then 0
 *         else the decimal number whose ordinary depiction is $this.rep]
 * </pre>
 *
 * @author David P & Zach B
 *
 */
public class NaturalNumber3 extends NaturalNumberSecondary {

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

    /**
     * Representation of {@code this}.
     */
    private String rep;

    /**
     * Creator of initial representation.
     */
    private void createNewRep() {
        // create new representation of @this.
        this.rep = "";
    }

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

    /**
     * No-argument constructor.
     */
    public NaturalNumber3() {
        // call create new rep and create no args constructor.
        this.createNewRep();
    }

    /**
     * Constructor from {@code int}.
     *
     * @param i
     *            {@code int} to initialize from
     */
    public NaturalNumber3(int i) {
        assert i >= 0 : "Violation of: i >= 0";

        this.createNewRep();
        // if i is greater than zero, set @this.rep to integer.tostring
        if (i > 0) {
            this.rep = Integer.toString(i);
        }
    }

    /**
     * Constructor from {@code String}.
     *
     * @param s
     *            {@code String} to initialize from
     */
    public NaturalNumber3(String s) {
        assert s != null : "Violation of: s is not null";
        assert s.matches("0|[1-9]\\d*") : ""
                + "Violation of: there exists n: NATURAL (s = TO_STRING(n))";

        // create a new representation
        this.createNewRep();
        // if s is not an empty string, set @this.rep to the string s.
        if (!s.equals("0")) {
            this.rep = s;
        }
    }

    /**
     * Constructor from {@code NaturalNumber}.
     *
     * @param n
     *            {@code NaturalNumber} to initialize from
     */
    public NaturalNumber3(NaturalNumber n) {
        assert n != null : "Violation of: n is not null";
        // create new representation
        this.createNewRep();
        // if n is not zero, we can set @this.rep to be whatever n is but as a string.
        if (!n.isZero()) {
            this.rep = n.toString();
        }
    }

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

    @Override
    public final NaturalNumber 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() {
        //create new rep to clear.
        this.createNewRep();
    }

    @Override
    public final void transferFrom(NaturalNumber source) {
        assert source != null : "Violation of: source is not null";
        assert source != this : "Violation of: source is not this";
        assert source instanceof NaturalNumber3 : ""
                + "Violation of: source is of dynamic type NaturalNumberExample";
        /*
         * This cast cannot fail since the assert above would have stopped
         * execution in that case.
         */
        NaturalNumber3 localSource = (NaturalNumber3) source;
        this.rep = localSource.rep;
        localSource.createNewRep();
    }

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

    @Override
    public final void multiplyBy10(int k) {
        assert 0 <= k : "Violation of: 0 <= k";
        assert k < RADIX : "Violation of: k < 10";

        // if @this.rep is not an empty string and k is not 0,
        // set @this.rep to int.tostring
        if (!(this.rep.equals("") && k == 0)) {
            this.rep += Integer.toString(k);
        }
    }

    @Override
    public final int divideBy10() {
        // initialize a final return type/digit.
        int finalDigit = 0;
        // if this.rep is not an empty string, set finalDigit to be the parsed
        // version of @this.rep from the beginning to the second to last digit.
        if (!this.rep.equals("")) {
            finalDigit = Integer
                    .parseInt(this.rep.substring(this.rep.length() - 1));
            this.rep = this.rep.substring(0, this.rep.length() - 1);
        }
        // return the final digit.
        return finalDigit;
    }

    @Override
    public final boolean isZero() {
        // return if @this.rep length is 0.
        return this.rep.length() == 0;
    }
}

```