```java
import components.naturalnumber.NaturalNumber;

/**
 * Controller class.
 *
 * @author Yakob Getu
 */
public final class NNCalcController1 implements NNCalcController {

    /**
     * Model object.
     */
    private final NNCalcModel model;

    /**
     * View object.
     */
    private final NNCalcView view;

    /**
     * Useful constants.
     */
    private static final NaturalNumber TWO = new NaturalNumber2(2),
            INT_LIMIT = new NaturalNumber2(Integer.MAX_VALUE);

    /**
     * Updates this.view to display this.model, and to allow only operations
     * that are legal given this.model.
     *
     * @param model
     *            the model
     * @param view
     *            the view
     * @ensures [view has been updated to be consistent with model]
     */
    private static void updateViewToMatchModel(NNCalcModel model,
            NNCalcView view) {

        //// Retrieve and display top and bottom numbers from the model.
        NaturalNumber top = model.top();
        NaturalNumber bottom = model.bottom();
        view.updateTopDisplay(top);
        view.updateBottomDisplay(bottom);

        //// Update UI controls based on codes logic and model conditions.
        view.updateRootAllowed(model.bottom().compareTo(INT_LIMIT) <= 0
                && model.bottom().compareTo(TWO) >= 0);
        view.updatePowerAllowed(model.bottom().compareTo(INT_LIMIT) <= 0);
        view.updateSubtractAllowed(model.bottom().compareTo(top) <= 0);
        view.updateDivideAllowed(!model.bottom().isZero());

    }

    /**
     * Constructor.
     *
     * @param model
     *            model to connect to
     * @param view
     *            view to connect to
     */
    public NNCalcController1(NNCalcModel model, NNCalcView view) {
        this.model = model;
        this.view = view;
        updateViewToMatchModel(model, view);
    }

    @Override
    public void processClearEvent() {
        /*
         * Get alias to bottom from model
         */
        NaturalNumber bottom = this.model.bottom();
        /*
         * Update model in response to this event
         */
        bottom.clear();
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processSwapEvent() {
        /*
         * Get aliases to top and bottom from model
         */
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        /*
         * Update model in response to this event
         */
        NaturalNumber temp = top.newInstance();
        temp.transferFrom(top);
        top.transferFrom(bottom);
        bottom.transferFrom(temp);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processEnterEvent() {
        // Copy the bottom number to the top in the model.
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        top.copyFrom(bottom);
        // Update the view to match the current state of the model.
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processAddEvent() {
        // Add the bottom number to the top and update both in the model.
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        top.add(bottom);
        bottom.transferFrom(top);
        // Refresh the view to reflect the updated model.
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processSubtractEvent() {
        // Subtract the bottom number from the top and update both.
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        top.subtract(bottom);
        bottom.transferFrom(top);
        // Refresh the view to reflect changes.
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processMultiplyEvent() {
        // Multiply the top number by the bottom and update both.
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        top.multiply(bottom);
        bottom.transferFrom(top);
        // Update the view to match the new model state.
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processDivideEvent() {
        // Divide the top number by the bottom, handle the remainder.
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        NaturalNumber remain = top.divide(bottom);
        bottom.transferFrom(top);
        top.transferFrom(remain);
        // Refresh the view with the new values.
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processPowerEvent() {
        // Raise the top number to the power of the bottom's integer value.
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        top.power(bottom.toInt());
        bottom.transferFrom(top);
        // Update the view to display the results of the power operation.
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processRootEvent() {
        // Calculate the root of the top number based on the bottom's value.
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        top.root(bottom.toInt());
        bottom.transferFrom(top);
        // Update the view to show the results of the root operation.
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processAddNewDigitEvent(int digit) {
        // Append a new digit to the bottom number.
        NaturalNumber bottom = this.model.bottom();
        bottom.multiplyBy10(digit);
        // Update the view to reflect the new number.
        updateViewToMatchModel(this.model, this.view);
    }

}

```