```java
import components.naturalnumber.NaturalNumber;

/**
 * Controller class.
 *
 * @author Shyam Sai Bethina
 */
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),
            ZERO = new NaturalNumber2();

    /**
     * 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) {

        //Gets the values from the top and bottom displays
        NaturalNumber input = model.bottom();
        NaturalNumber output = model.top();

        //Updates the bottom and top display based model values
        view.updateBottomDisplay(input);
        view.updateTopDisplay(output);

        /*
         * If the input is greater than 0, then the divide
function is allowed,
         * but if it is 0 or less than 0, the divide button
becomes disabled to
         * avoid divide by 0 and negative numbers
         */
        if (!input.isZero()) {
            view.updateDivideAllowed(true);
        } else {
            view.updateDivideAllowed(false);
        }

        /*
         * If the input is less than 0, then the subtract function
is allowed,
         * but if it less than the output, the subtract button
becomes disabled
         * to avoid negative numbers
         */
        if (input.compareTo(output) < 0) {
            view.updateSubtractAllowed(true);
        } else {
            view.updateSubtractAllowed(false);
        }

        /*
         * If the input is less than the integer limit, then the
power function
         * is allowed, but if it greater than the integer limit,
the power
         * button becomes disabled to avoid outOfBounds error
because the
         * processPowerMethod uses the toInt(), which cannot
convert any object
         * that is greater than the integer limit
         */
        if (input.compareTo(INT_LIMIT) < 0) {
```

```java
            view.updatePowerAllowed(true);
        } else {
            view.updatePowerAllowed(false);
        }

        /*
         * If the input is greater than or equal to two, and is less than the
         * integer limit, then the root function is allowed. But if it greater
         * than the integer limit or is less than two, the power button becomes
         * disabled to avoid outOfBounds error because the processRootMethod
         * uses the toInt(), which cannot convert any object that is greater
         * than the integer limit
         */
        if (input.compareTo(TWO) >= 0 && input.compareTo(INT_LIMIT) < 0) {
            view.updateRootAllowed(true);
        } else {
            view.updateRootAllowed(false);
        }
    }

    /**
     * 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() {
        /*
         * Get aliases to top and bottom from model
         */
        NaturalNumber bottom = this.model.bottom();
        NaturalNumber top = this.model.top();
        /*
         * Update model in response to this event
         */
        top.copyFrom(bottom);
        /*
```

```
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processAddEvent() {
        /*
         * 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
         */
        bottom.add(top);
        top.clear();
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

    @Override
    public void processSubtractEvent() {
        /*
         * 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
         */
        top.subtract(bottom);
        bottom.transferFrom(top);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

    @Override
```

```java
    public void processMultiplyEvent() {
        /*
         * 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
         */
        top.multiply(bottom);
        bottom.transferFrom(top);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processDivideEvent() {
        /*
         * Get aliases to top and bottom from model
         */
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        /*
         * remainder is the remainder when top is divide by bottom
         */
        NaturalNumber remainder = top.divide(bottom);
        /*
         * Update model in response to this event, top displays
 the remainder
         */
        bottom.transferFrom(top);
        top.transferFrom(remainder);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processPowerEvent() {
        /*
         * Update model in response to this event
         */
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        /*
         * bottom is converted to an integer, and top becomes top to the power
         * of bottom
         */
        top.power(bottom.toInt());
        /*
         * Update model in response to this events
         */
        bottom.transferFrom(top);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

    @Override
    public void processRootEvent() {
        /*
         * Update model in response to this event
         */
        NaturalNumber top = this.model.top();
        NaturalNumber bottom = this.model.bottom();
        /*
         * bottom is converted to an integer, and top becomes bottom root of top
         */
        top.root(bottom.toInt());
        /*
         * Update model in response to this events
         */
        bottom.transferFrom(top);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }

    @Override
    public void processAddNewDigitEvent(int digit) {
```

```java
        /*
         * Update model in response to this event
         */
        NaturalNumber bottom = this.model.bottom();
        /*
         * Adds a new digit to the input by multiplying by 10 and adding the new
         * digit
         */
        bottom.multiplyBy10(digit);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);
    }

}

```