```
import components.naturalnumber.NaturalNumber;
import components.naturalnumber.NaturalNumber2;

/**
 * Controller class.
 *
 * @author Shafin Alam
 */
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.
     *
```

```java
     * @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) {
        //Save the top and bottom of the model to NaturalNumbers
        NaturalNumber top = model.top();
        NaturalNumber bottom = model.bottom();
        /*
         * Should check if subtraction is allowed, (bottom must be smaller than
         * top). Otherwise it's allowed.
         */
        if (bottom.compareTo(top) > 0) {
            view.updateSubtractAllowed(false);
        } else {
            view.updateSubtractAllowed(true);
        }
        /*
         * Should check if division is allowed, (division by 0 = illegal).
         * Otherwise it's allowed.
         */
        if (bottom.isZero()) {
            view.updateDivideAllowed(false);
        } else {
            view.updateDivideAllowed(true);
        }
        /*
         * Should check if power is allowed, (bottom can't be larger than the
         * maximum of any integer). Otherwise it is allowed.
         */
        if (bottom.compareTo(INT_LIMIT) <= 0) {
            view.updatePowerAllowed(true);
        } else {
            view.updatePowerAllowed(false);
        }
        /*
         * Should check if root is allowed, (bottom must be larger than 2).
         * Otherwise it is allowed.
         */
        if (bottom.compareTo(TWO) >= 0 && bottom.compareTo(INT_LIMIT) <= 0) {
            view.updateRootAllowed(true);
        } else {
            view.updateRootAllowed(false);
        }
        /*
         * View should change accordingly.
         */
        view.updateTopDisplay(top);
        view.updateBottomDisplay(bottom);

    }

    /**
     * Constructor.
     *
     * @param model
```

```java
     *            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() {
        /*
         * Should set the top equal to the bottom or copy it. Update model in
         * response to this event
         */
        this.model.top().copyFrom(this.model.bottom());
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

    @Override
    public void processAddEvent() {
        /*
         * Should add the top and the bottom, and then clear the top; saving the
         * answer to the bottom. Update model in response
```

```
to this event
         */
        this.model.bottom().add(this.model.top());
        this.model.top().clear();
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

    @Override
    public void processSubtractEvent() {
        /*
         * Should subtract the bottom from the top, and
then save that number to
         * the bottom; afterwards, clearing the top.
Update model in response to
         * this event
         */
        this.model.top().subtract(this.model.bottom());
        this.model.bottom().copyFrom(this.model.top());
        this.model.top().clear();
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

    @Override
    public void processMultiplyEvent() {
        /*
         * Should multiply the top and bottom together,
and clear the top;
         * saving the answer to the bottom. Update model in response to this
         * event
         */
        this.model.bottom().multiply(this.model.top());
        this.model.top().clear();
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

    @Override
    public void processDivideEvent() {
        /*
         * Should divide the top by the bottom, and save the answer to the
         * bottom, clearing the top afterwards. Update model in response to this
         * event
         */
        NaturalNumber r = this.model.top().divide(this.model.bottom());
        this.model.bottom().copyFrom(this.model.top());
        this.model.top().copyFrom(r);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

```

```java
    @Override
    public void processPowerEvent() {
        /*
         * Should take the top to the power of the bottom, and save the answer
         * to the bottom, clearing the top afterwards. Update model in response
         * to this event
         */
        this.model.top().power(this.model.bottom().toInt());
        this.model.bottom().copyFrom(this.model.top());
        this.model.top().clear();
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

    @Override
    public void processRootEvent() {
        /*
         * Should take the bottom"th" root of the top, saving the answer to the
         * bottom, clearing the top afterwards. Update model in response to this
         * event
         */
        this.model.top().root(this.model.bottom().toInt());
        this.model.bottom().copyFrom(this.model.top());
        this.model.top().clear();
        /*
```

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

    }

    @Override
    public void processAddNewDigitEvent(int digit) {
        /*
         * This will update the bottom when the user inputs a number, by
         * multiplying by 10, and then adding the digit. Update model in
         * response to this event
         */
        this.model.bottom().multiplyBy10(digit);
        /*
         * Update view to reflect changes in model
         */
        updateViewToMatchModel(this.model, this.view);

    }

}

```