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

/**
 * Controller class.
 *
 * @author Gabe Azzarita
 */
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) {

        NaturalNumber top = model.top();
        NaturalNumber bottom = model.bottom();

        boolean rootAllowed = false;
        boolean subtractAllowed = false;
        boolean powerAllowed = false;
        boolean divideAllowed = false;

        // For root method, r root must be int and <= 2
        if (bottom.compareTo(TWO) >= 0 && bottom.compareTo(INT_LIMIT) <= 0) {
            rootAllowed = true;
        }
```

```
        // Cannot have a subtraction that results in a negative
        if (bottom.compareTo(top) <= 0) {
            subtractAllowed = true;
        }
        // For power method, p power must be int
        if (bottom.compareTo(INT_LIMIT) <= 0) {
            powerAllowed = true;
        }
        // Cannot divide by 0
        if (!bottom.isZero()) {
            divideAllowed = true;
        }

        view.updateTopDisplay(top);
        view.updateBottomDisplay(bottom);
        view.updateRootAllowed(rootAllowed);
        view.updateSubtractAllowed(subtractAllowed);
        view.updatePowerAllowed(powerAllowed);
        view.updateDivideAllowed(divideAllowed);
    }

    /**
     * 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() {

    NaturalNumber top = this.model.top();
    NaturalNumber bottom = this.model.bottom();

    top.transferFrom(bottom);
    updateViewToMatchModel(this.model, this.view);
}

@Override
public void processAddEvent() {

    NaturalNumber top = this.model.top();
    NaturalNumber bottom = this.model.bottom();

    top.add(bottom);
    bottom.transferFrom(top);
    updateViewToMatchModel(this.model, this.view);
}

@Override
public void processSubtractEvent() {

    NaturalNumber top = this.model.top();
    NaturalNumber bottom = this.model.bottom();

    top.subtract(bottom);
    bottom.transferFrom(top);
    updateViewToMatchModel(this.model, this.view);
}
```

```
@Override
public void processMultiplyEvent() {

    NaturalNumber top = this.model.top();
    NaturalNumber bottom = this.model.bottom();

    top.multiply(bottom);
    bottom.transferFrom(top);
    updateViewToMatchModel(this.model, this.view);
}

@Override
public void processDivideEvent() {

    // We need to set top = remainder and bottom = quotient
    NaturalNumber top = this.model.top();
    NaturalNumber bottom = this.model.bottom();
    NaturalNumber remainder = top.divide(bottom);

    bottom.transferFrom(top);
    top.transferFrom(remainder);
    updateViewToMatchModel(this.model, this.view);
}

@Override
public void processPowerEvent() {

    NaturalNumber top = this.model.top();
    NaturalNumber bottom = this.model.bottom();

    top.power(bottom.toInt());
    bottom.transferFrom(top);
    updateViewToMatchModel(this.model, this.view);
}

@Override
public void processRootEvent() {

    NaturalNumber top = this.model.top();
    NaturalNumber bottom = this.model.bottom();

    top.root(bottom.toInt());
    bottom.transferFrom(top);
    updateViewToMatchModel(this.model, this.view);
}

@Override
public void processAddNewDigitEvent(int digit) {

    NaturalNumber bottom = this.model.bottom();
```

```
        bottom.multiplyBy10(digit);

        updateViewToMatchModel(this.model, this.view);
    }
}
```