```java
public class Cistern
{
    /**
     * Volume maximum en litres.
     */
    private double maximumVolume;

    /**
     * Volume actuel en litres.
     */
    private double currentVolume;

    /**
     * Constructeur : définit une nouvelle citerne par ses dimensions
     * extérieures.
     *
     * @param pRadius le rayon de la citerne en m
     * @param pHeight la hauteur de la citerne en m
     */
    public Cistern(double pRadius, double pHeight)
    {
        maximumVolume = Math.PI * pRadius * pRadius * pHeight * 1000; // Attention à la conversion m^3 -> litres
        currentVolume = 0;
    }

    /**
     * Ajouter de l'eau à la citerne jusqu'à faire le plein.
     *
     * @param pVolume volume d'eau à ajouter
     */
    public void add(double pVolume)
    {
        if (currentVolume + pVolume <= maximumVolume)
        {
            currentVolume = currentVolume + pVolume;
        }
        else
        {
            currentVolume = maximumVolume;
        }
    }

    /**
     * Retirer de l'eau de la citerne jusqu'à la vider.
     *
     * @param pVolume volume d'eau à retirer
     */
    public void drain(double pVolume)
    {
        if (pVolume <= currentVolume)
        {
            currentVolume = currentVolume - pVolume;
        }
        else
        {
            currentVolume = 0;
        }
    }

    /**
     * Ajouter de l'eau à la citerne jusqu'à faire le plein.
     *
     * @param pVolume volume d'eau à ajouter
     * @return true si l'action a été effectuée sans déborder
     */
    public boolean addOK(double pVolume)
    {
        if (currentVolume + pVolume <= maximumVolume)
        {
            currentVolume = currentVolume + pVolume;
            return true;
        }
        else
        {
            currentVolume = maximumVolume;
            return false;
        }
    }

    /**
     * Retirer de l'eau de la citerne jusqu'à la vider .
     *
     * @param pVolume volume d'eau à retirer
     * @return true si tout le volume recquis a pu être retiré
     */
    public boolean drainOK(double pVolume)
    {
        if (pVolume <= currentVolume)
        {
            currentVolume = currentVolume - pVolume;
            return true;
        }
        else
        {
            currentVolume = 0;
            return false;
        }
    }
```

```java
    /**
     * Retourner le volume actuel .
     *
     * @return volume actuel d'eau dans la citerne
     */
    public double getCurrentVolume()
    {
        return currentVolume;
    }

    /**
     * Retourner le volume maximal .
     *
     * @return volume maximal d'eau dans la citerne
     */
    public double getMaximumVolume()
    {
        return maximumVolume;
    }

    /**
     * Retourner le taux de remplissage actuel .
     *
     * @return taux de remplissage en pourcentage
     */
    public double getCurrentRate()
    {
        return currentVolume / maximumVolume * 100;
    }

    /**
     * Retourner un message indiquant le volume et le taux de remplissage
     * actuels .
     *
     * @return texte contenant le message d'information
     */
    public String toString()
    {
        return "Fill level : " + currentVolume + "l / " + maximumVolume + "l (" + getCurrentRate() + "%)";
    }
}
```

```java
public class MainFrame extends javax.swing.JFrame
{
     private Cistern cistern = new Cistern(1, 2);

     public MainFrame()
     {
          initComponents();
          int max = (int) cistern.getMaximumVolume();
          volumeSlider.setMaximum(max);
          volumeSlider.setMajorTickSpacing(max / 10); // Afficher uniquement 10 ticks (lignes)
          updateView();
     }

     public double round2Decimals(double v)
     {
          // pour expliquer le principe
          return ((int) (v * 100)) / 100.0;
     }

     public double roundDecimals(double v, int nbrDecimals)
     {
          double p = Math.pow(10, nbrDecimals);
          return ((int) (v * p)) / p;
     }

     public void updateView()
     {
          currentVolumeLabel.setText(roundDecimals(cistern.getCurrentVolume(), 2) + " litres");
          currentRateLabel.setText(roundDecimals(cistern.getCurrentRate(), 2) + " %");
          currentRateProgressBar.setValue((int) cistern.getCurrentRate());
     }
// Skipped: ... initComponents { ... }
    private void addButtonActionPerformed(java.awt.event.ActionEvent evt)//GEN-FIRST:event_addButtonActionPerformed
    {//GEN-HEADEREND:event_addButtonActionPerformed
          double v = Double.valueOf(volumeTextField.getText());
          cistern.add(v);
          updateView();
    }//GEN-LAST:event_addButtonActionPerformed

    private void drainButtonActionPerformed(java.awt.event.ActionEvent evt)//GEN-FIRST:event_drainButtonActionPerformed
    {//GEN-HEADEREND:event_drainButtonActionPerformed
          cistern.drain(Double.valueOf(volumeTextField.getText()));
          updateView();
    }//GEN-LAST:event_drainButtonActionPerformed

     private void volumeSliderStateChanged(javax.swing.event.ChangeEvent evt)//GEN-FIRST:event_volumeSliderStateChanged
     {//GEN-HEADEREND:event_volumeSliderStateChanged
          volumeTextField.setText(String.valueOf(volumeSlider.getValue()));
     }//GEN-LAST:event_volumeSliderStateChanged
// Skipped: ... Look & Feel
     // Variables declaration - do not modify//GEN-BEGIN:variables
     private javax.swing.JButton addButton;
     private javax.swing.JLabel currentRateLabel;
     private javax.swing.JProgressBar currentRateProgressBar;
     private javax.swing.JLabel currentVolumeLabel;
     private javax.swing.JButton drainButton;
     private javax.swing.JLabel jLabel1;
     private javax.swing.JLabel jLabel2;
     private javax.swing.JSlider volumeSlider;
     private javax.swing.JTextField volumeTextField;
     // End of variables declaration//GEN-END:variables
}
```