

```

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 m3 -> 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 requis a pu être retiré
     */
    public boolean drainOK(double pVolume)
    {
        if (pVolume <= currentVolume)
        {
            currentVolume = currentVolume - pVolume;
            return true;
        }
        else
        {
            currentVolume = 0;
            return false;
        }
    }
}

```

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

/**
 * 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() + "%)";
}
}
```

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

    public MainFrame()
    {
        initComponents();
        updateView();
    }

    public void updateView()
    {
        currentVolumeLabel.setText(((int) (cistern.getCurrentVolume() * 100) / 100.0) + " litres");
        currentRateLabel.setText(((int) (cistern.getCurrentRate() * 100) / 100.0) + "%");
        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
// 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.JTextField volumeTextField;
    // End of variables declaration//GEN-END:variables
}
```