R2.02	Séance 1	2021-2022

<u>Remarque préliminaire</u> : cet énoncé est par nature incomplet, il invite à prendre conscience de l'importance de la documentation de l'API Java pour résoudre les problèmes posés, et a pour but d'accompagner la découverte de la programmation d'interfaces graphiques en Java.

*Bien qu'il soit demandé d'écrire du code Java de façon incrémentale, en modifiant le code au fur et à mesure des questions posées, il reste fortement conseillé d'archiver chacune des réponses sous forme d'un fichier de sauvegarde. Les codes sources et les réponses aux différentes questions devront être déposés sur Moodle (**cf. consignes en fin d'énoncé à respecter scrupuleusement**).*

A. Hello World

1. Recopier le code source fourni ci-dessous (également disponible sur Moodle), le compiler et l'exécuter.

```
import javax.swing.*;

public class HelloWorldSwing extends JFrame {

    /**
     * Program entry point
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                // Create the frame and show it
                HelloWorldSwing frame = new HelloWorldSwing();
                frame.pack();
                frame.setVisible(true);
            }
        });
    }

    /** Initialize the HelloWorldSwing frame components. */
    public HelloWorldSwing() {
        setTitle("HelloWorldSwing");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Add the ubiquitous "Hello World" label.
        JLabel label = new JLabel("Hello World");
        add(label);
    }
}
```

2. Combien de classes ont été définies dans le programme ci-dessus ? (conseil : on pourra regarder les fichiers .class générés lors de la compilation du code source)
   Que contient chacune de ces classes ? (conseil : on pourra simplifier la méthode *main(…)* par un appel direct aux 3 lignes de code contenues dans la méthode *run()*)

3. En s'appuyant sur la Javadoc, décrire précisément le fonctionnement du programme (pour chaque ligne du code source).

4. Toujours en s'appuyant sur la Javadoc, modifier le programme pour que le message affiché dans la fenêtre soit passé en paramètre au programme (par l'intermédiaire de la variable *String[] args*). Vérifier le bon fonctionnement du nouveau programme.

5. Étendre maintenant le programme pour qu'il puisse prendre plusieurs messages en paramètre, affichés chacun dans un label différent (conseil : on dupliquera alors autant que nécessaire les 2 lignes permettant l'ajout du label). Observer le résultat fourni par le programme et comparer à celui attendu.

6. Ajouter l'instruction `getContentPane().setLayout(new java.awt.FlowLayout());` avant la création des objets *JLabel* et relancer le programme pour s'assurer de son bon fonctionnement. En s'appuyant sur la Javadoc des méthodes *getContentPane()* et *setLayout(…)* et de la classe *FlowLayout*, expliquer l'intérêt de cette instruction.

B. Celsius Converter

1. Recopier le code source fourni ci-dessous (également disponible sur Moodle), le compiler et l'exécuter.

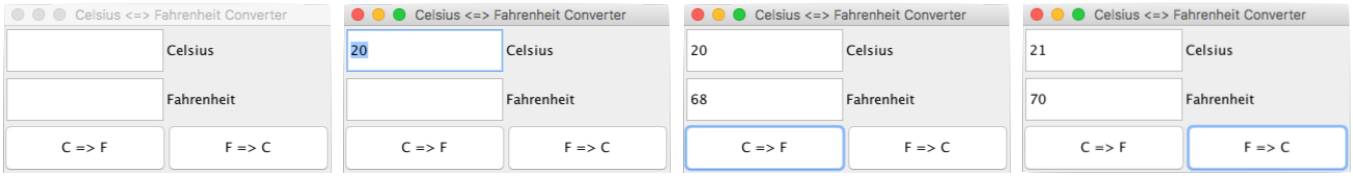
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CelsiusConverterGUI extends JFrame {

    private JLabel celsiusLabel;
    private JButton convertButton;
    private JLabel fahrenheitLabel;
    private JTextField tempTextField;

    /**
     * Program entry point
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                CelsiusConverterGUI frame = new CelsiusConverterGUI();
                frame.pack();
                frame.setVisible(true);
            }
        });
    }

    /** Initialize the CelsiusConverterGUI frame components. */
    public CelsiusConverterGUI() {

        setTitle("Celsius Converter");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        tempTextField = new JTextField();
        celsiusLabel = new JLabel();
        convertButton = new JButton();
        fahrenheitLabel = new JLabel();

        celsiusLabel.setText("Celsius");
        convertButton.setText("Convert");
        convertButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                convertButtonActionPerformed(evt);
            }
        });
        fahrenheitLabel.setText("Fahrenheit");

        getContentPane().setLayout(new GridLayout(2,2));

        add(tempTextField);
        add(celsiusLabel);
        add(convertButton);
        add(fahrenheitLabel);
    }

    /** Parse degrees Celsius as a double and convert to Fahrenheit */
    private void convertButtonActionPerformed(ActionEvent evt) {
        int tempFahr = (int)((Double.parseDouble(tempTextField.getText())) * 1.8 + 32);
        fahrenheitLabel.setText(tempFahr + " Fahrenheit");
    }
}
```

2. Recenser dans ce programme les différents composants graphiques utilisés. En consultant la Javadoc, expliquer brièvement l'intérêt de chacune des classes utilisées.

3. La disposition des composants dans la fenêtre est différente de celle adoptée dans le premier programme. Expliquer pourquoi. À quoi sert la classe *GridLayout* ? (conseil : se référer une fois encore à la Javadoc).

4. Identifier le code qui est exécuté lors du clic sur le bouton. Comment ce code a-t-il été associé au bouton ?

5. **Question complémentaire, si vous avez le temps (facultatif) :** Faire évoluer le programme pour qu'on puisse effectuer la conversion des températures dans les deux sens, tel qu'illustré par l'exemple graphique ci-dessous.



Remettre individuellement les codes sources des versions finales des deux exemples (fichiers .java) ainsi que les réponses aux questions (document pdf). **La remise devra s'effectuer à la fin de la séance.**