

Licences Sciences Technologies Santé

Semestre 4

Rapport complet du projet Java :

Développement d'un outil graphique de

gestion de projet par graphes

(GraphTracer).

Auteurs :

Abdelaziz HOCINE (22212597)

Melissa IKERROUYENE (22315882)

Formation :

**Licence Sciences, Technologies, Santé – Informatique – 2e
année**

Année universitaire :

2024-2025



Remerciements

Nous remercions nos enseignants et encadrants pour leurs conseils et leur disponibilité tout au long du projet, ainsi que nos camarades pour les échanges constructifs.



Résumé :

Ce rapport présente le développement de l'application Java **GraphTracer**, un outil graphique permettant de modéliser et de suivre l'avancement d'un projet sous forme de graphe orienté. L'utilisateur peut créer, modifier, supprimer et déplacer des tâches (sommets) et des dépendances (arcs), sauvegarder et charger des graphes, et visualiser le chemin critique. Le projet met en œuvre les principes de la programmation orientée objet et propose une interface graphique intuitive.

Mots-clés : gestion de projet, graphe orienté, Java, interface graphique, programmation objet.

Sommaire :

1. Introduction
2. Contexte et objectifs
3. Conception et architecture logicielle
4. Réalisation et méthodes
5. Bibliographie
6. Annexes
7. Résultats
8. Perspectives d'évolution
9. Conclusion

Glossaire :

- **Graphe orienté** : Structure composée de sommets (tâches) reliés par des arcs (dépendances) orientés.
- **Tâche** : Unité de travail dans un projet, représentée par un sommet.
- **Arc** : Dépendance ou relation de précédence entre deux tâches.
- **Sérialisation** : Processus de sauvegarde d'un objet Java dans un fichier.
- **Chemin critique** : Ensemble de tâches déterminant la durée minimale nécessaire à la réalisation complète du projet. Tout retard sur l'une de ces tâches retarde l'ensemble du projet.

1. Introduction :

Le suivi de l'avancement d'un projet complexe nécessite des outils adaptés pour visualiser les dépendances entre tâches et identifier les points critiques. Ce rapport présente le développement de l'application **GraphTracer**, réalisée en binôme dans le cadre du module Programmation Orientée Objet, L'objectif est de fournir un outil graphique interactif pour la gestion de projets, basé sur la modélisation par graphes orientés.

La suite du document détaille le contexte, la conception, la réalisation, les résultats obtenus et les perspectives d'évolution.

2. Contexte et objectifs :

2.1. Contexte :

La gestion de projet implique la planification, le suivi et la coordination de multiples tâches interdépendantes. Les graphes orientés sont un modèle pertinent pour représenter ces dépendances. L'outil développé vise à faciliter la visualisation et la modification dynamique de ces graphes.

2.2. Objectifs :

- Permettre la création, modification, suppression et déplacement de tâches et de liens.
- Permettre la suppression automatique des arcs reliés à un sommet lors de sa suppression.
- Assurer la sauvegarde et le chargement des graphes.
- Visualiser le chemin critique.
- Offrir une interface graphique intuitive et ergonomique.
- Respecter les principes de la programmation orientée objet (modularité, encapsulation, abstraction).

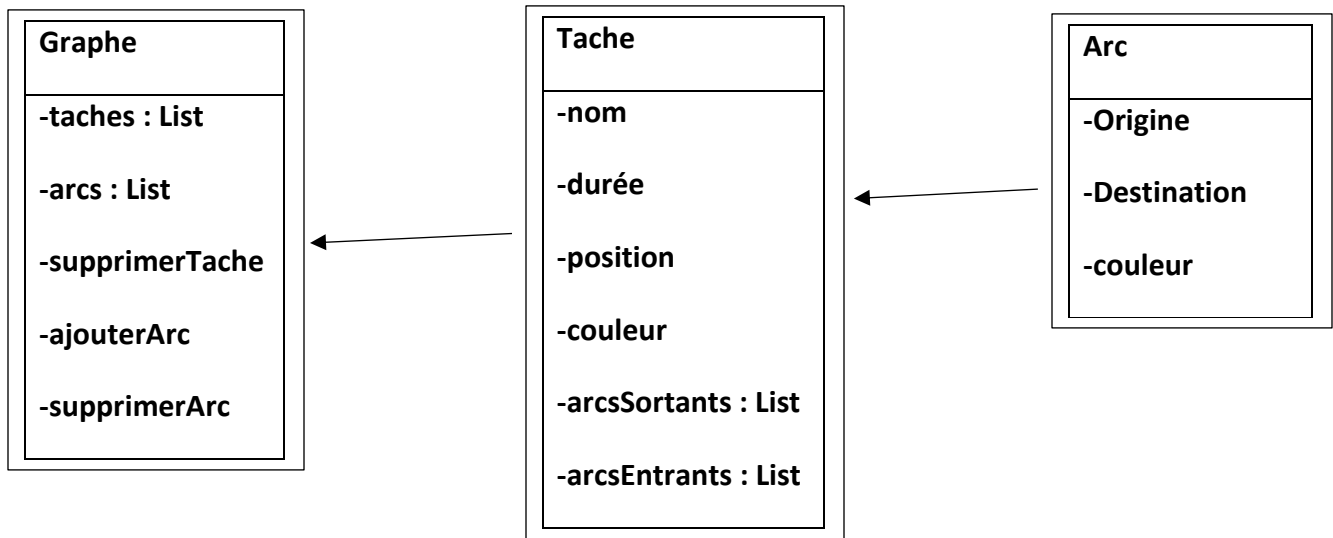
3. Conception et architecture logicielle :

3.1. Structure générale :

L'application est structurée autour de plusieurs classes principales :

- **Graphe** : Modélise l'ensemble des tâches et des arcs.
- **Tache** : Représente une tâche du projet.
- **Arc** : Représente une dépendance entre deux tâches.
- **ZoneDeGraphe** : Panneau graphique pour l'affichage et l'interaction.
- **EditeurDeGraphe** : Fenêtre principale d'édition.
- **Constantes** : Centralise les paramètres de l'application.
- **ChargeurDeGraphe (interface)** : Définit les méthodes pour le chargement de graphes.

3.2. Diagramme de classes (extrait) :



3.3. Principes de conception :

- **Modularité** : chaque classe a une responsabilité unique.
- **Encapsulation** : les attributs sont privés, accessibles via [getters/setters](#).
- **Extensibilité** : l'interface `ChargeurDeGraphe` permet d'ajouter d'autres modes de chargement.
- **Gestion des exceptions** : protection contre les opérations interdites (ex : suppression des tâches spéciales).

4. Réalisation et méthodes :

4.1. Fonctionnalités principales :

- **Ajout/suppression/modification de tâches** (nom, durée, couleur, position)
- **Création/suppression de liens** (arcs orientés)
- **Déplacement graphique des tâches** (en déplaçant la tâche avec la souris)
- **Sélection visuelle** (mise en évidence des éléments sélectionnés – l’entourer en vers)
- **Sauvegarde/chargement** (fichier.ser)
- **Affichage du chemin critique** (tâches entourées en rouge)
- **Impression du graphe** (via le menu)

4.2. Méthodes et algorithmes :

- **Ajout d’une tâche** : vérification du nom, création, ajout à la liste, rafraîchissement graphique.
- **Suppression d’une tâche** : suppression en cascade des arcs associés.
- **Ajout d’un arc** : vérification de la validité, ajout, mise à jour graphique.

- **Déplacement** : gestion du drag via événements souris.
- **Sérialisation** : sauvegarde et chargement via `ObjectOutputStream` et `ObjectInputStream`.
- **Affichage** : dessin des tâches (cercles) et arcs (lignes + flèches) dans `ZoneDeGraphe`.

4.3. Quelques problèmes rencontrés et les solutions :

- ✗ **Gestion des tâches spéciales** : L'ajout automatique de "début" et "fin" via la méthode publique provoquait une exception.
✓ **Solution** : ajout direct dans la liste interne.
- ✗ **Sélection graphique** : Difficulté à sélectionner des arcs proches.
✓ **Solution** : calcul de la distance au centre de l'arc avec un seuil.
- ✗ **Synchronisation** : Suppression des arcs liés lors de la suppression d'une tâche.
✓ **Solution** : création d'un tableau pour conserver les sommets précédents (prédécesseurs) pour chaque tâche (sommet), afin d'identifier et supprimer automatiquement tous les arcs dont elle est l'origine ou la destination.
- ✗ **Robustesse** : Gestion des erreurs utilisateur (noms invalides, suppression interdite, etc...).

5. Bibliographie :

5.1 Bibliothèques standards utilisées :

- **javax.swing**

Pour l'interface graphique (fenêtres, boutons, menus, dialogues, panneau de dessin, etc.)

Exemples : JFrame, JPanel, JMenuBar, JMenuItem, JOptionPane, JColorChooser.

- **java.awt**

Pour les composants graphiques de bas niveau, la gestion des couleurs, des formes, des événements souris, etc.

Exemples : Graphics, Color, Point, Container, SpringLayout, MouseEvent, MouseAdapter, MouseMotionAdapter, AffineTransform.

- **java.awt.print**

Pour la gestion de l'impression graphique (impression du graphe).

- **javax.print.attribute**

Pour les attributs d'impression (format papier, nom du job, etc.).

- **java.util**

Pour les collections (List, ArrayList), la gestion des listes de tâches, d'arcs, etc...

- **java.io**

Pour la gestion des fichiers (sauvegarde/chargement du graphe).

5.2 Résumé des imports principaux dans le projet :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.AffineTransform;
import java.awt.print.*;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.ArrayList;
import javax.print.attribute.*;
import javax.print.attribute.standard.*;
```

5.3 Remarques :

- Aucune bibliothèque externe n'est utilisée (tout est standard Java).
- Principalement **Swing** est utilisé pour l'interface graphique et **AWT** est utilisé pour le dessin et la gestion des événements.
- Les packages javax.print et java.awt.print sont utilisés pour la fonctionnalité d'impression.

5.4 Liens et ressources utiles :

- [Documentation officielle Java SE](#)
- [Tutoriel Java Swing](#)
- [AWT \(Abstract Windows Toolkit\)](#)

(Ctrl + clic-gauche) sur le lien pour l'ouvrir

- Tutoriels YouTube (exemples) :



5.5 Environnement de développement utilisé :

- [Eclipse IDE](#)

6. Annexes techniques :

6.1 Structure du projet :

- **Encapsulation** : attributs privés, accès via getters/setters.

Exemple : dans la classe Taches.java :

```
package graphtracer;

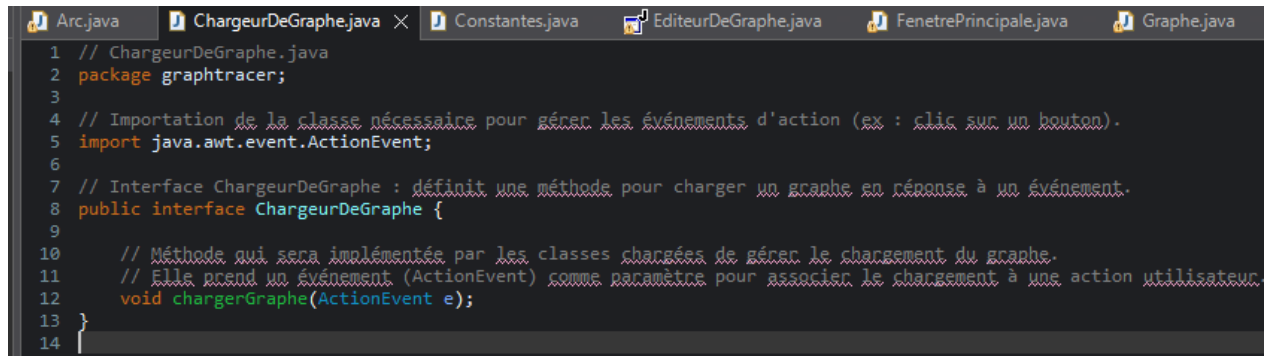
// Classe représentant une tâche dans un graphe de dépendances
public class Tache implements Serializable {
    private String nom;
    private long duree;
    private int px, py;
    private Color couleur;
    private List<Arc> arcsSortants;
    private List<Arc> arcsEntrants;
    private List<String> tachesachevées;
    private int dateAuPlusTot;
    private int dateAuPlusTard;

    // Constructeur de la tâche
    public Tache(String nom, long duree, int px, int py) {
    }

    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }
    public long getDuree() { return duree; }
    public void setDuree(long duree) { this.duree = duree; }
    public int getPx() { return px; }
    public void setPx(int px) { this.px = px; }
    public int getPy() { return py; }
    public void setPy(int py) { this.py = py; }
    public Color getCouleur() { return couleur; }
    public void setCouleur(Color couleur) { this.couleur = couleur; }
    public List<Arc> getArcsSortants() { return arcsSortants; }
    public List<Arc> getArcsEntrants() { return arcsEntrants; }
    public int getDateAuPlusTot() { return dateAuPlusTot; }
    public void setDateAuPlusTot(int date) { this.dateAuPlusTot = date; }
    public int getDateAuPlusTard() { return dateAuPlusTard; }
    public void setDateAuPlusTard(int date) { this.dateAuPlusTard = date; }
}
```

- **Extensibilité** : interface `ChargeurDeGraphe` pour futures extensions.

La classe `ChargeurDeGraphe.java` :



```

1 // ChargeurDeGraphe.java
2 package graphtracer;
3
4 // Importation de la classe nécessaire pour gérer les événements d'action (ex : clic sur un bouton).
5 import java.awt.event.ActionEvent;
6
7 // Interface ChargeurDeGraphe : définit une méthode pour charger un graphe en réponse à un événement.
8 public interface ChargeurDeGraphe {
9
10     // Méthode qui sera implémentée par les classes chargées de gérer le chargement du graphe.
11     // Elle prend un événement (ActionEvent) comme paramètre pour associer le chargement à une action utilisateur.
12     void chargerGraphe(ActionEvent e);
13 }
14

```

- **Robustesse** : gestion des erreurs, protection des tâches spéciales.

La classe `Graphe.java` :



```

1 package graphtracer;
2
3 import java.awt.BasicStroke;
4
5 public class Graphe implements Serializable {
6     private List<Tache> taches;
7     private List<Arc> arcs;
8
9     public Graphe() {
10         taches = new ArrayList<>();
11         arcs = new ArrayList<>();
12         ajouterTachesSpeciales();
13     }
14
15     private void ajouterTachesSpeciales() {
16         ajouterTache(new Tache("début", 0, 100, 100));
17         ajouterTache(new Tache("fin", 0, 500, 300));
18     }
19 }
20

```

- **Interface intuitive** : sélection visuelle, menus clairs, retour utilisateur.

- **Persistence** : sauvegarde/chargement simples via sérialisation.

La classe `EditeurDeGraphe.java` :

```

16 // Sauvegarde le graphe dans un fichier
17 private void sauvegarderGraphe() {
18     JFileChooser fileChooser = new JFileChooser();
19     if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
20         File file = fileChooser.getSelectedFile();
21         try {
22             graphe.sauvegarder(file.getAbsolutePath());
23         } catch (IOException ex) {
24             JOptionPane.showMessageDialog(this,
25                 "Erreur lors de la sauvegarde : " + ex.getMessage(),
26                 "Erreur",
27                 JOptionPane.ERROR_MESSAGE);
28         }
29     }
30 }
31 }
32
33 // Charge un graphe à partir d'un fichier
34 public void chargerGraphe() {
35     JFileChooser fileChooser = new JFileChooser();
36     if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
37         File file = fileChooser.getSelectedFile();
38         try {
39             Graphe grapheCharge = Graphe.charger(file.getAbsolutePath());
40             chargerGraphe(grapheCharge);
41         } catch (Exception ex) {
42             JOptionPane.showMessageDialog(this,
43                 "Erreur lors du chargement : " + ex.getMessage(),
44                 "Erreur",
45                 JOptionPane.ERROR_MESSAGE);
46         }
47     }
48 }
49 }

```

- **Impression** : possibilité d'imprimer la vue graphique (fonction print)

La classe `EditeurDeGraphe.java` :

```

// Permet d'imprimer le graphe en ajustant son échelle à la page
private void imprimerGraphe() {
    PrinterJob job = PrinterJob.getPrinterJob();
    job.setJobName("Impression du graphe");

    PrintRequestAttributeSet attributes = new HashPrintRequestAttributeSet();
    attributes.add(MediaSizeName.ISO_A4);
    attributes.add(new JobName("GraphTracer - Impression", null));

    if (job.printDialog(attributes)) {
        try {
            job.print(attributes); // Lancement de l'impression
        } catch (PrinterException e) {
            JOptionPane.showMessageDialog(this,
                "Erreur lors de l'impression : " + e.getMessage(),
                "Erreur d'impression",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}

```


6.2 Exécution et utilisation avec l'interface graphique :

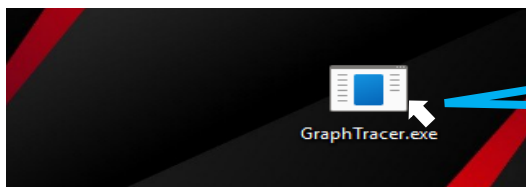
- **Démarrage** : lancement de la classe GraphTracer qui ouvre le tableau de bord.

On ouvre le dossier GraphTracer :

```
GraphTracer/  
├─ GraphTracer.exe # Exécutable Windows  
├─ README.txt # Documentation  
├─ graphtracer/  
│   ├── Graphe.java # Modèle de graphe  
│   ├── Tache.java # Représentation des taches  
│   ├── Arc.java # Représentation des arcs  
│   └─ ... # Autres composants  
├─ img/ # Ressources graphiques  
│   ├── graphe_oriente.png # Exemple de graphe orienté  
│   ├── graphe_non_or.png # Exemple de graphe non orienté  
│   └─ logo.png # Logo de l'application
```

En utilisant le fichier exécutable GraphTracer.exe :

Clic-droite sur le programme Graphtracer.exe puis cliquer sur ouvrir



Ou bien en utilisant la terminale :

On ouvre le dossier **GraphTracer** dans la terminale puis on entre les commandes suivantes :

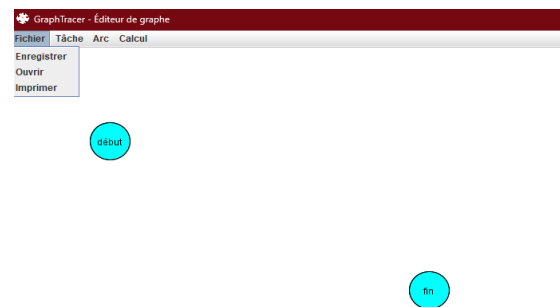
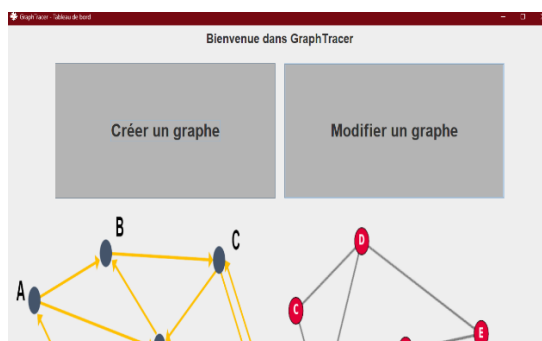
```
> javac graphtracer/*.java
> java graphtracer/GraphTracer
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

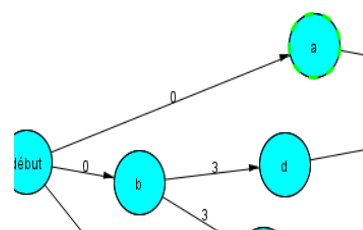
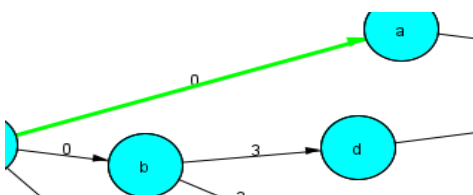
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\abdel\Desktop\java\Projet_java\GraphTracer> javac graphtracer/*.java
PS C:\Users\abdel\Desktop\java\Projet_java\GraphTracer> java graphtracer/GraphTracer
PS C:\Users\abdel\Desktop\java\Projet_java\GraphTracer> |
```

- **Création d'un graphe** : bouton « Créer un graphe » crée un nouveau graphe avec tâches spéciales.
- **Modification** : On a un bouton « Modifier un graphe » pour charger un graphe déjà enregistré sur l'appareil et un menu pour ajouter, supprimer, renommer tâches et arcs.



- **Sélection** : clic sur tâche ou arc pour sélectionner, indication visuelle.



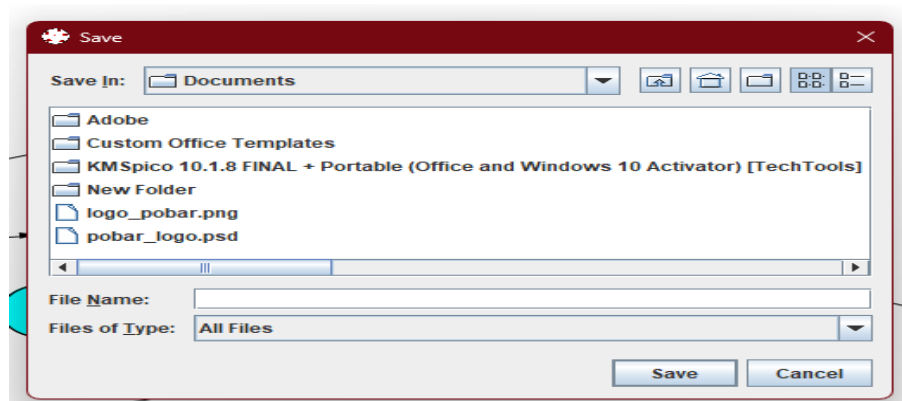
- **Déplacement** : drag & drop des tâches pour repositionner.
- **Sauvegarde/chargement** : via menus Fichier, avec dialogues.

```

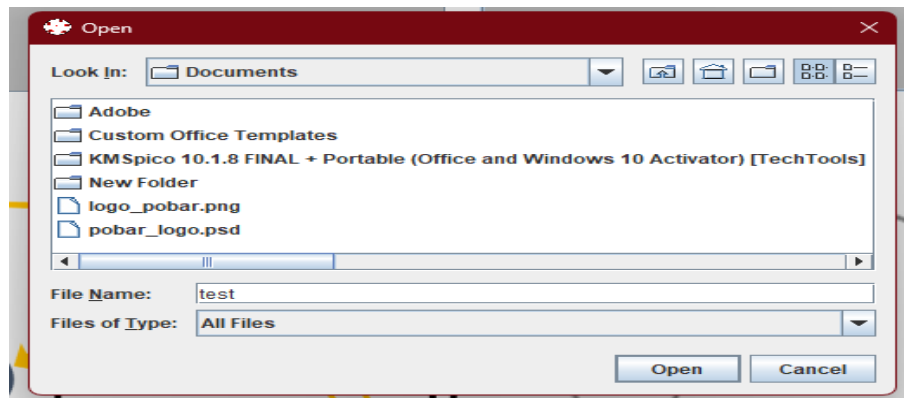
16 // ...
17 // Sauvegarde le graphe dans un fichier
18 private void sauvegarderGraphe() {
19     JFileChooser fileChooser = new JFileChooser();
20     if (fileChooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
21         File file = fileChooser.getSelectedFile();
22         try {
23             graphe.sauvegarder(file.getAbsolutePath());
24         } catch (IOException ex) {
25             JOptionPane.showMessageDialog(this,
26                 "Erreur lors de la sauvegarde : " + ex.getMessage(),
27                 "Erreur",
28                 JOptionPane.ERROR_MESSAGE);
29         }
30     }
31 }
32
33 // Charge un graphe à partir d'un fichier
34 public void chargerGraphe() {
35     JFileChooser fileChooser = new JFileChooser();
36     if (fileChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
37         File file = fileChooser.getSelectedFile();
38         try {
39             Graphe grapheCharge = Graphe.charger(file.getAbsolutePath());
40             chargerGraphe(grapheCharge);
41         } catch (Exception ex) {
42             JOptionPane.showMessageDialog(this,
43                 "Erreur lors du chargement : " + ex.getMessage(),
44                 "Erreur",
45                 JOptionPane.ERROR_MESSAGE);
46         }
47     }
48 }
49 // ...

```

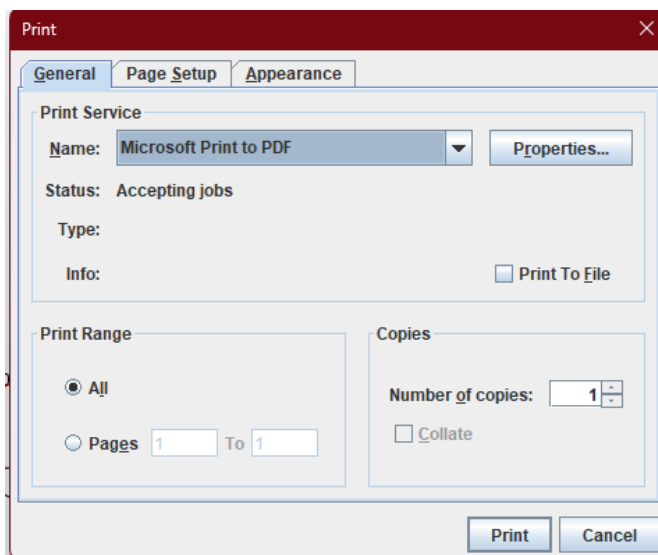
La fenêtre 'Save' est pour sauvegarder un graphe elle est gérée et appelée par la méthode sauvegarderGraphe().



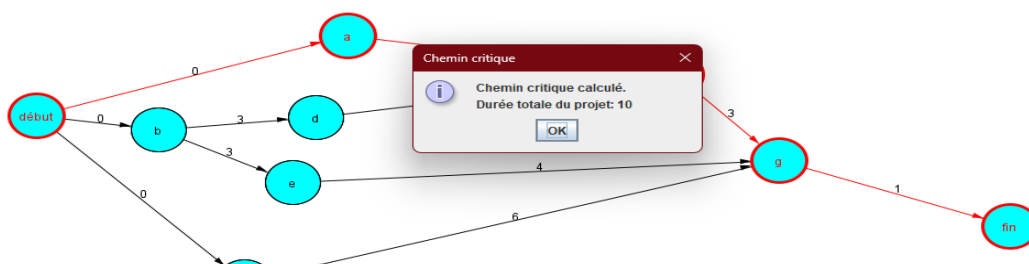
La fenêtre 'Open' est pour ouvrir un graphe déjà sauvegardé sur la machine, elle est gérée et appelée par la méthode chargerGraphe().



- **Impression** : option dans le menu pour imprimer la zone graphique.



- **Affichage** : tâches critiques entourées en rouge, durées affichées sur arcs.



7. Résultats :

L'application fonctionne conformément à l'énoncé de projet :

- Interface graphique réactive et intuitive.
- Toutes les fonctionnalités principales sont implémentées.
- Code structuré, commenté, facilement extensible.
- Tests réalisés sur différents scénarios (création, suppression, sauvegarde, déplacement).
- Gestion robuste des erreurs et des cas particuliers.

8. Perspectives d'évolution :

Calcul automatique du chemin critique

Intégrer un calcul dynamique du chemin critique, mis à jour en temps réel à chaque modification du graphe, pour offrir à l'utilisateur une analyse instantanée de la planification.

Export du graphe en PDF

Ajouter la possibilité d'exporter le graphe au format PDF, afin de faciliter le partage, l'archivage et l'intégration dans des rapports professionnels.

9. Conclusion :

Ce projet a permis de mettre en œuvre une application complète de gestion de graphes orientés en Java, avec une interface graphique interactive et une architecture orientée objet solide. Les choix techniques garantissent modularité, extensibilité et robustesse. Le travail en binôme a favorisé la qualité du code et la bonne organisation.