

Introduction

Le but de ce TP est de vous apprendre à manipuler l'environnement de développement Eclipse afin de coder du début à la fin une application de gestion de planning simplifiée.

Notation du TP

À la fin de chaque séance vous aurez à remettre un certain nombre de documents afin d'en évaluer la qualité.

Ces documents seront à envoyer à l'adresse : nohri.graoudi@gmail.com en précisant dans l'intitulé du courriel la lettre de votre groupe.

Un seul courriel pour tout le groupe suffira.

La notation se basera sur les critères suivants :

- la progression durant la séance dans l'écriture du programme;
- le lancement, le bon fonctionnement et la stabilité du programme rendu;
- le respect du cahier des charges;
- la cohérence de la documentation donnée aux endroits où elle est judicieuse.

Important : le code doit être documenté en respectant les règles de la **Javadoc**.

La note globale sera sur 30.

Cahier des charges de l'application

Éléments attendus

L'application sera codée en Java.

L'application sera persistante à l'aide de la bibliothèque JDBC permettant le support de différents

SGBD (Système de Gestion de Base de Données). La sauvegarde se fera à chaque modification effectuée dans l'application. A l'ouverture de l'application, la sauvegarde sera rechargée pour afficher l'état au moment de la fermeture.

La fenêtre principale possédera une barre de menus avec pour certains des menus des sous-menus :

Fichier	Édition	Vue	Aide
> Quitter	> Créer	> Liste	
	> Éditer	> Semaine	
	> Marquer	> Mois	
	> Dupliquer		
	> Supprimer		

Les actions dans la fenêtre principal ainsi que dans le menu « Edition » sont :

- Créer : permet de créer une nouvelle tâche. La fenêtre de création de tâche s'ouvre alors.

- Éditer : permet d'éditer la tâche actuellement sélectionnée. Cette action n'est pas disponible si aucune tâche n'est sélectionnée de même que si plus d'une tâche sont sélectionnées.
- Marquer : permet de marquer la ou les tâches sélectionnées comme effectuées. Ce ou ces tâches deviennent grisées. Cette action n'est pas disponible si aucune tâche n'est sélectionnée.
- Dupliquer : permet de dupliquer une tâche sélectionnée. Cette action n'est pas disponible si aucune tâche n'est sélectionnée de même que si plus d'une tâche sont sélectionnées.
- Supprimer : permet de supprimer la ou les tâches sélectionnées. Cette action n'est pas disponible si aucune tâche n'est sélectionnée.

La fenêtre principale proposera un menu déroulant permettant de choisir le mode d'affichage parmi les 3 modes « Liste », « Semaine » et « Mois ». Ces 3 modes sont également sélectionnables dans le menu « Vue ».

La fenêtre principal afficher a également en bas à droite la date du jour.

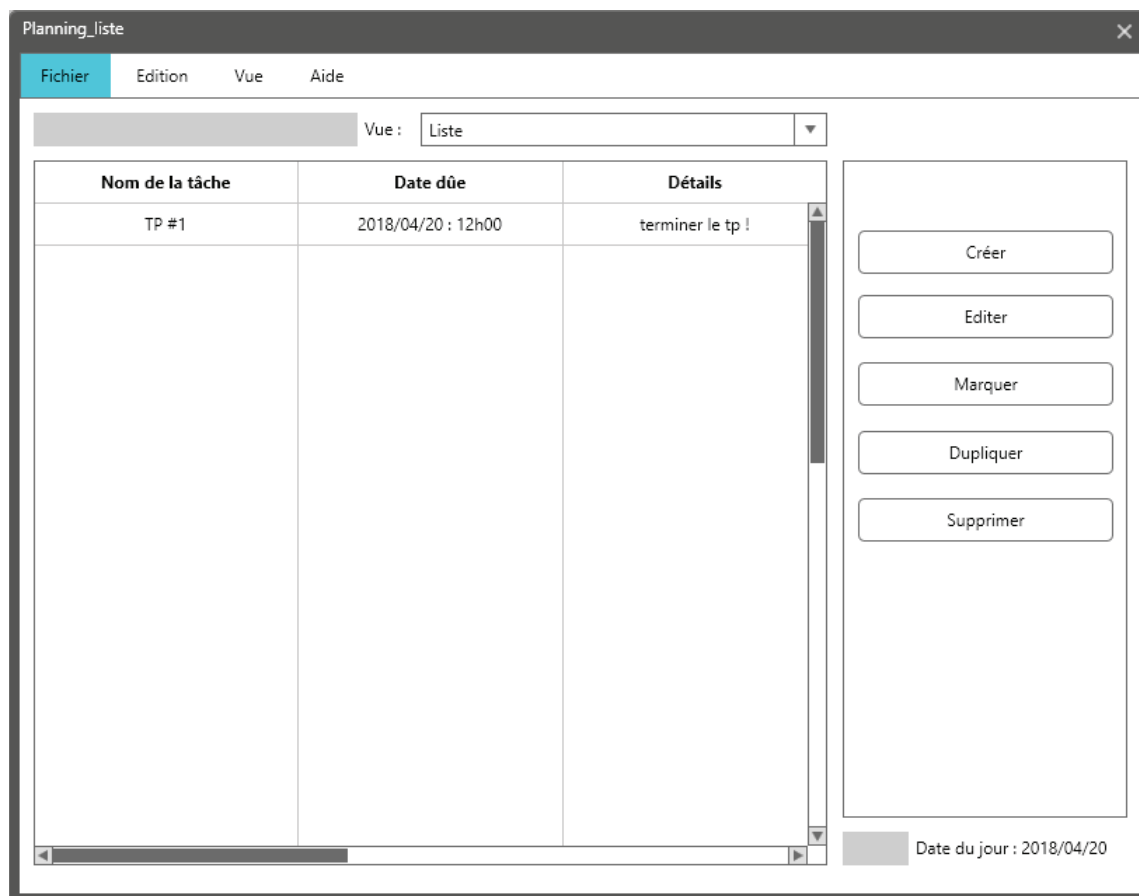
La fenêtre de création d'une tâche contiendra les champs suivants :

- « Nom de la tâche » : le nom de la tâche affichée
- « Date due » : la date, et l'heure (optionnelle) à laquelle est due la tâche. Un calendrier permettra une sélection simple de la date dans un format aaaa/mm/jj. L'heure sera entrée au format sur 24h.
- « Détails » : un zone de texte permettant de décrire de manière précise la tâche.
- Un bouton OK pour accepter la tâche (et la sauvegarder).
- Un bouton Annuler pour annuler l'action en cours.

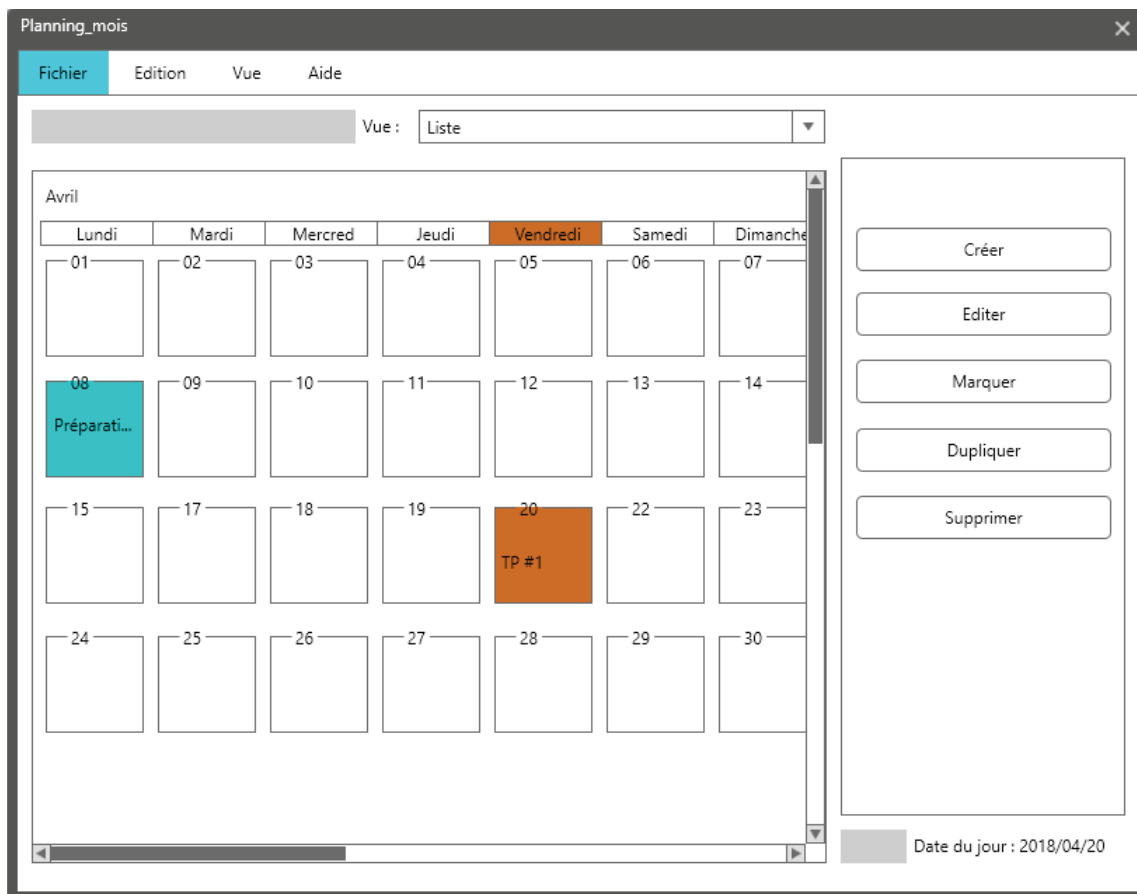
Cette fenêtre servira également afin d'éditer ou de dupliquer une tâche.

Mockups

Vue du programme en mode « Liste »



Vue du programme en mode « Mois »



Vue de la fenêtre de création, édition et clone d'une tâche.

Nouvelle tâche

Nom de la tâche :

TP #1

Date d'ue :

2018/04/20 : 12h00

Détails :

Donec augue enim, volutpat at ligula et, dictum laoreet sapien. Sed maximus feugiat tincidunt. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nulla eu mollis leo, eu elementum nisl. Curabitur cursus turpis nibh, egestas efficitur diam tristique non.

◀ Mar 2014 ▶

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

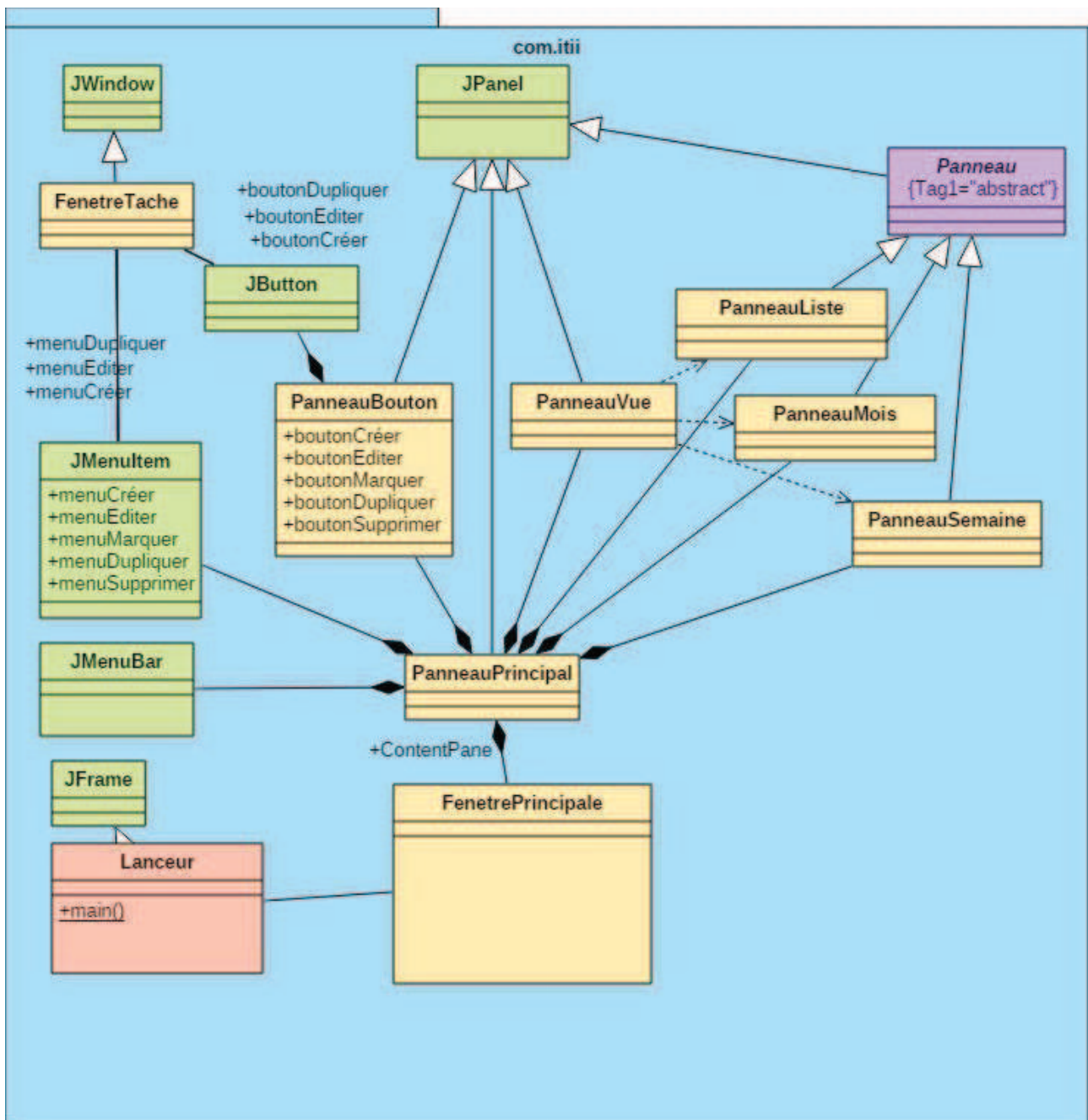
Annuler

OK

Diagramme des classes

La diagramme des classes ci-dessous présente l'ensemble des classes dont vous aurez besoin ainsi que leurs interactions.

La structure des packages est laissée libre, mais une certaine logique doit émaner de votre arborescence de classes (les noms des classes sont donnés en français. Privilégiez plutôt l'anglais pour leur nom dans votre programme).



TP#1

Eclipse

Vous devriez déjà avoir un environnement fonctionnel et opérationnel sachant que nous avons déjà codé quelques « hands on » durant le cours.

Dans cette première séance, nous allons apprendre à utiliser Eclipse et à programmer notre première application en respectant un certain nombre de règles.

Le premier projet

Au cours de ce projet, vous aurez à suivre un cahier des charges strict vous permettant, étapes après étapes, de coder votre application de planning.

Pour débiter, créez un nouveau projet en allant dans le menu **File → New → Java Project**.

Nommez ce projet en fonction de la lettre de votre groupe de la manière suivante : **ITII_2018_#** (où # est à remplacer par votre lettre de groupe).

Vérifiez que **Java 8** est correctement sélectionnée et que le reste est bien par défaut comme sur la capture ci-contre (capture Illustration 1: Création de projet).

Cliquez sur **Finish**.

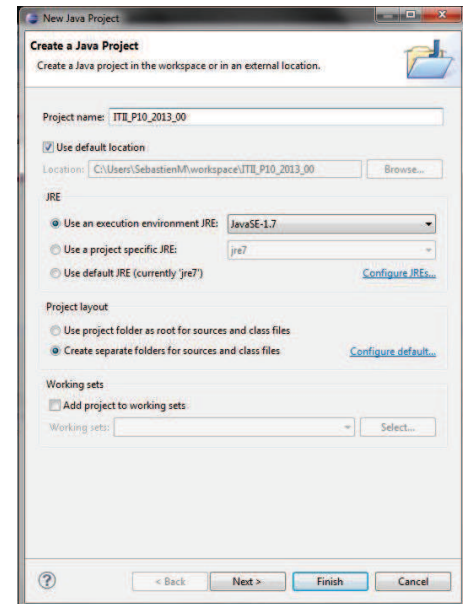


Illustration 1: Création de projet

Vérifiez que vous êtes sur la perspective **Debug** comme nous l'avons vu dans le document **cours_java_2017-2018_preparation.pdf** avec l'ensemble des vues nécessaires.

La liste des perspectives est visible en haut à droite de Eclipse (Capture Illustration 2: Perspectives).

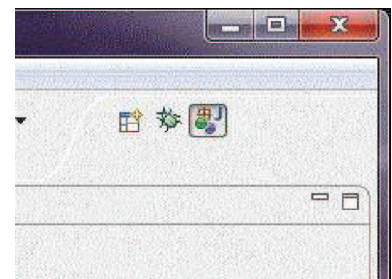


Illustration 2: Perspectives

Voici le détail de ces vues :

- Dans la vue de gauche :
 - **Package Explorer** : permet de visualiser tous vos projets, paquets, bibliothèques, fichiers sources etc...
 - **Navigator** : permet d'explorer tout le dossier de votre projet.
- Dans la vue d'en bas :
 - **Problems** : liste les erreurs et messages d'alertes liés à votre code
 - **Console** : montre les messages de sortie lors de l'exécution de votre application

S'il vous manque des vues, allez dans le menu **Windows → Show View** et sélectionnez les vues désirées. Il est possible ensuite de mettre en forme les différentes vues en faisant un glisser/déposer de ces vues vers l'endroit de votre IDE où vous souhaitez les placer.

Ces vues nous seront indispensables pour la suite de cette séance de mise en pratique.

Dans la vue **Projet Explorateur** déroulez l'arbre de votre projet pour faire apparaître le répertoire **src**.

Ce répertoire, comme son nom l'indique, est là pour accueillir toutes les sources de votre projet, c'est à dire l'ensemble des fichiers *.java.

Il est possible d'y créer des paquetages (ou packages en anglais) qui vous permettront de classer vos fichiers sources de manière logique. Nous y reviendrons.

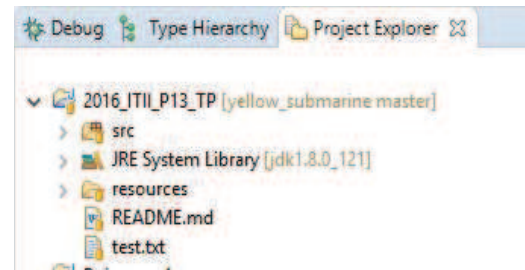


Illustration 3: Répertoire "src"

Notez que l'ensemble du projet se situe par défaut dans le répertoire :

C:\Users\'votre_compte'\eclipse-workspace\ITII_2018_#

Packages

Commençons par créer notre premier *paquetage* que nous nommerons **com.itii**. Ce paquetage est le paquetage par défaut de tout notre projet.

Pour se faire, cliquez sur le répertoire **src** avec le bouton droit de la souris puis **New → Package**.

Une nouvelle boîte de dialogue s'ouvre.

Entrez comme nom de package **com.itii** puis cliquez sur **Finish** (capture Illustration 4: Nouveau package com.itii).

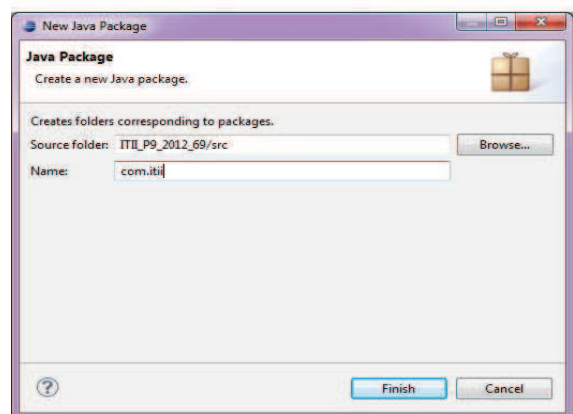


Illustration 4: Nouveau package com.itii

Pour rappel, un package se nomme toujours en minuscule.

Ce package apparaît alors dans votre vue **Package Explorer** bel et bien sous la forme d'un package. Maintenant, si vous regardez dans la vue **Navigator**, vous remarquerez qu'en réalité Eclipse ne fait que créer une série de répertoires **com\itii**. On constate ici l'une des différences majeures entre les 2 vues **Package Explorer** et **Navigator**.

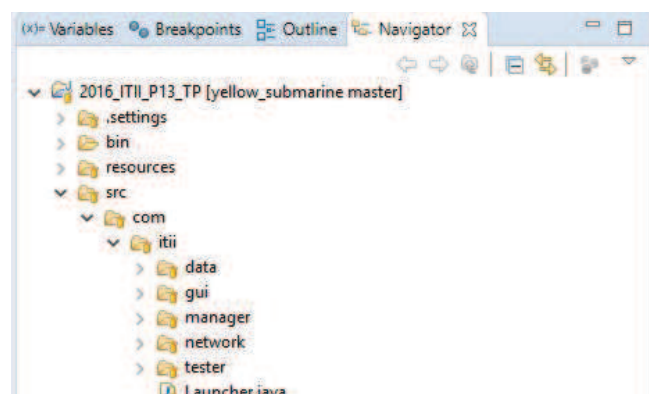


Illustration 5: Vue "Navigator"

Launcher

Nous allons créer le package **com.itii.planning** (ou rajouter le package **planning** au package **com.itii** existant). Il contiendra l'ensemble du code relatif à la partie graphique de notre application.

Notre package `com.itii.planning` est pour l'instant vide ; il ne contient aucune classe et apparaît sous forme d'une icône blanche. Nous allons maintenant créer notre toute première classe, mais pas des moindres, puisque c'est notre classe principale. Cette classe contiendra notre méthode **main** statique.

Faites un clic-droit sur le paquetage que nous venons juste de créer depuis la vue **Package Explorer**, et faites **New → Class** ; nommez cette nouvelle classe **Launcher**.

Pour rappel, il faut respecter le nommage d'une classe :

- ;le nom d'une classe commence toujours par une majuscule
- le nom du fichier et le nom de la classe doivent être identiques.

Laissez les autres champs par défaut puis cliquez sur **Finish** ; La classe s'ouvre alors dans l'éditeur sous le nom `Launcher.java` (capture Illustration 6: Première classe).

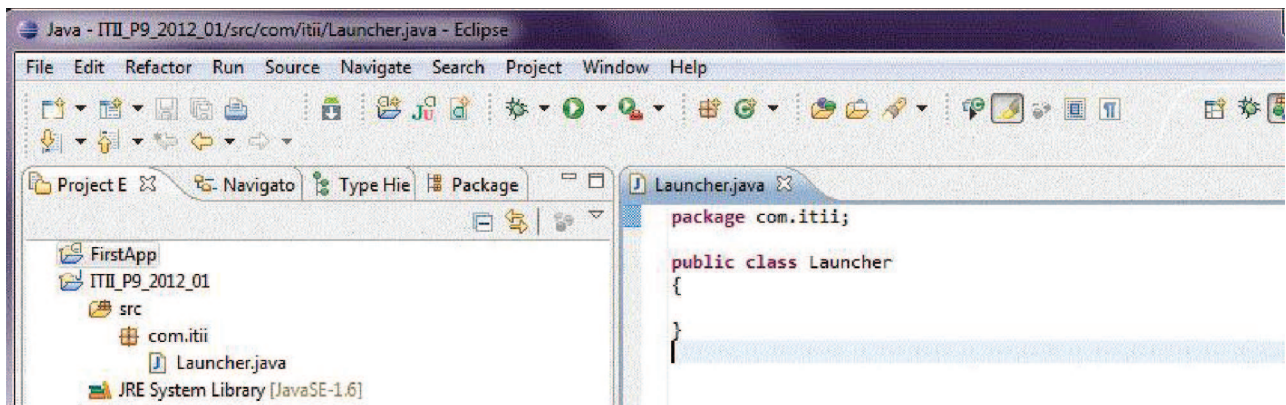


Illustration 6: Première classe

Le package **com.itii.planning** n'étant désormais plus vide, la couleur de l'icône du package passe de blanche à orangée.

Main

A vous de jouer. Maintenant que le squelette de la classe est créé, écrivez la première méthode indispensable à tout programme, à savoir, la méthode **Main**, selon ce qui a été vu en cours.

Astuce : Il existe un raccourci pour écrire cette méthode facilement et sans se tromper. Lorsque vous êtes dans l'éditeur, à l'intérieur de votre classe **Launcher** (entre les accolades), tapez au clavier le début du nom de la méthode main, à savoir «**ma**» et faites '**ctrl + espace**'. Ce raccourci vous permet de compléter automatiquement votre méthode (capture Illustration 7: Complétion automatique). Elle permet en plus de lister l'ensemble des méthodes de la classe.

Appuyez sur **entrée** pour sélectionner la première méthode **main**.

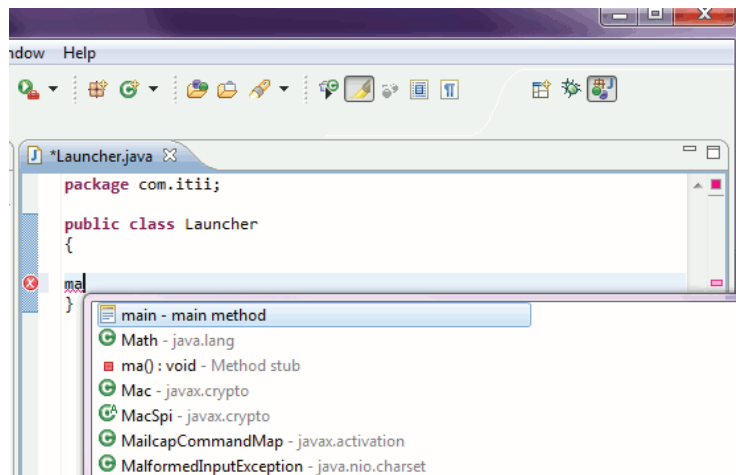


Illustration 7: Complétion automatique

Raccourcis: complétion automatique: Ctrl + Espace

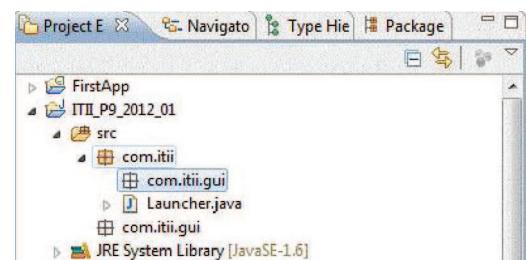
Maintenant que nous avons notre méthode **main**, nous allons créer une classe pour gérer la partie **interface graphique**.

Fenêtre principale

Cette classe, que nous appellerons **MainWindow (Fenêtre Principale)**, correspondra à la fenêtre principale de notre application. Avant de créer cette classe, nous allons créer un paquetage nommé **gui** appartenant au paquetage **com.itii.planning**. Faites un clic-droit sur le package **com.itii.planning** et créez ce nouveau package **com.itii.planning.gui**.

Il apparaît en blanc dans la vue **Project Explorer** (Illustration 8: package **com.itii.planning.gui**) puisqu'il est actuellement vide.

Créez la classe **MainWindow** appartenant à ce nouveau package, mais cette fois-ci nous allons la faire hériter de **JFrame** afin de l'afficher sous forme d'une fenêtre, avec barre de titre.

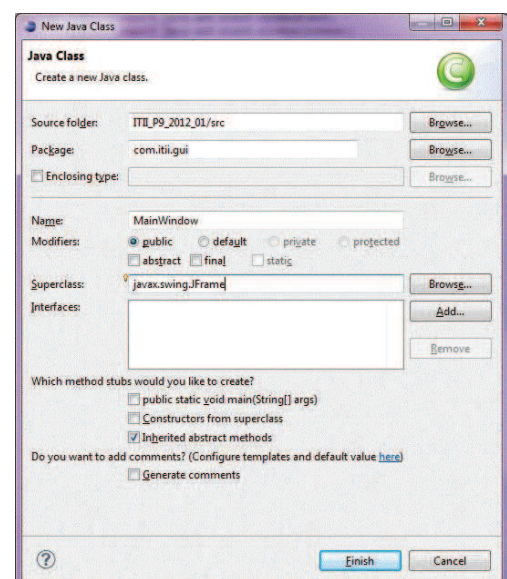


*Illustration 8: package **com.itii.planning.gui***

Pour ce faire, au moment de la création de la classe **MainWindow**, vous pouvez spécifier dans le champ **SuperClass** (capture Illustration 9: **MainWindow** héritant de **JFrame**) la valeur **JFrame** (en remplacement de **java.lang.Object**).

Lorsque vous avez tapé **JFrame** dans le champ de **SuperClass**, tout en restant dans ce champ, tapez **Ctrl + Espace** (comme nous l'avons vu juste précédemment) pour importer directement le package contenant cette classe **Jframe**.

L'entrée **JFrame** est alors résolue par Eclipse et remplacée



*Illustration 9: **MainWindow** héritant de **JFrame***

par son import : **javax.swing.JFrame**

Cliquez sur **Finish**.

Cette classe **MainWindow** sera considérée comme un **Singleton**. En effet, nous ne souhaitons pas que plusieurs instances de **MainWindow** existent. Le constructeur de cette classe devra donc respecter les règles du Singleton vues en cours.

Toujours dans notre classe **MainWindow**, nous allons ajouter une méthode d'initialisation **initialize()** que nous appellerons depuis son constructeur afin d'initialiser notre **fenêtre** (illustration ci-contre).

```
private MainWindow()  
{  
    initialize();  
}
```

*Illustration 10:
Constructeur avec
méthode d'initialisation*

L'initialisation consistera à spécifier une taille par défaut à notre **fenêtre** et à la rendre visible (car par défaut, la fenêtre n'est pas rendue visible).

Les 2 méthodes qui nous intéressent ici sont

- **setSize(...)**
- **setVisible(...)**

Utilisez ces 2 méthodes pour initialiser notre **MainWindow**.

Rappelons qu'en utilisant Ctrl + Espace, nous obtenons la liste complète des méthodes disponibles dans notre **MainWindow**. Il est alors possible d'obtenir des informations sur le fonctionnement de ses méthodes ; c'est ce qu'on appelle la Javadoc (capture Illustration 11: javadoc d'une méthode sélectionnée). L'autre moyen d'afficher la Javadoc est d'avoir le pointeur de souris sur la méthode, ou d'appuyer sur F2.

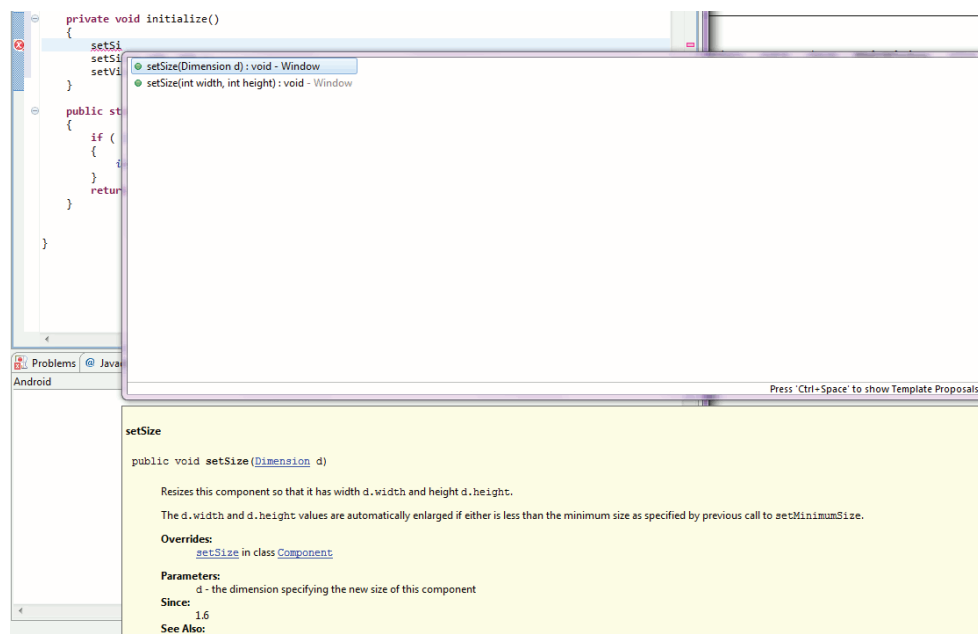


Illustration 11: javadoc d'une méthode sélectionnée

Raccourcis: appuyer sur F2 sur une méthode pour afficher sa Javadoc

Nous ajouterons à la fin l'appel aux méthodes «**validate()**» et «**repaint()**» afin de mettre à jour la liste complète des composants de notre fenêtre et de les redessiner au lancement du programme.

L'illustration 12 montre le squelette de la méthode « initialize ».

Notre classe **MainWindow** possède désormais une taille et sera visible. Il nous reste à l'afficher.

```

    }

    private void initialize()
    {
        setSize( 1, 1 );
        setVisible( true );

        this.validate();
        this.repaint();
    }

```

Illustration 12: Squelette de la méthode "Initialize"

L'affichage se fera en instanciant notre **MainWindow** depuis le *main* de la classe **Launcher**. Rappelons que **MainWindow** est un singleton et que son instantiation passe par l'appel à une méthode appelée communément «**getInstance()**». L'illustration ci-contre montre comment instancier ce singleton depuis le *main*.

```

public static void main ( String[] args )
{
    MainWindow.getInstance();
}

```

Illustration 13: Instanciation du Singleton MainWindow

Cependant, il faut importer le paquetage contenant **MainWindow** pour que la classe **Launcher** en ait connaissance.

Sous **Eclipse**, en faisant **Ctrl + Espace** juste après avoir tapé **MainWindow**, Eclipse importe alors automatiquement les bibliothèques manquantes.

L'autre moyen consiste à les importer à l'aide de : **Ctrl + Shift + O**.

Raccourcis : mise à jour des « imports » de paquetages : Ctrl + Shift + O

Maintenant que l'import est résolu, juste après avoir tapé **MainWindow**, ajoutez un point ".". Eclipse vous propose alors l'ensemble des méthodes disponibles dans **MainWindow**. Sélectionnez "**getInstance()**" pour récupérer l'instance unique de notre **MainWindow** (notre singleton). L'illustration ci-dessous vous montre comment Eclipse propose la « complétion » :

```

public static void main ( String[] args )
{
    MainWindow.

```

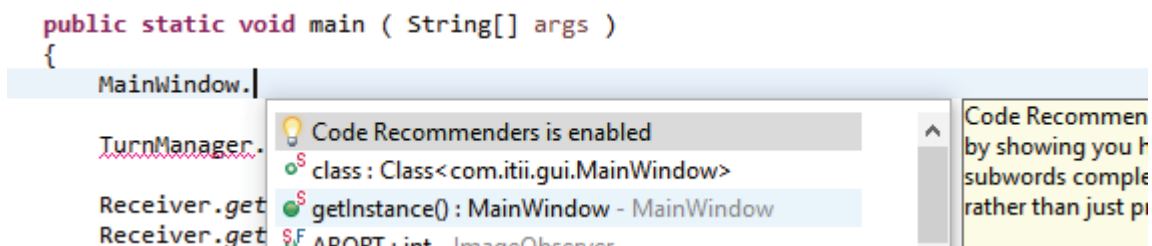


Illustration 14: Complétion automatique

Pensez à sauvegarder régulièrement votre travail. Soit en sauvegardant la classe actuellement ouverte, soit en sauvegardant l'intégralité de votre projet. Les 2 moyens sont donnés ci-après.

Raccourcis : sauvegarder la classe actuelle : Ctrl + S

Tout sauvegarder : Ctrl + Shift + S

Remarque : Un fichier qui n'est pas sauvegardé comporte une astérisque. Dans la capture ci-dessous on constate que **GridDisplay** n'est pas sauvegardé à l'inverse de **Coordinates** qui l'est.

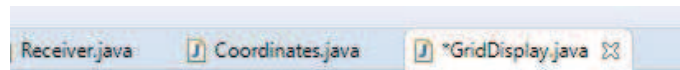


Illustration 15: Fichiers non sauvegardés

Avant de lancer notre programme, nous allons exécuter le raccourci **Ctrl + Shift + F**. Vous devriez constater quelque chose.

1- Expliquez

Lancement du programme

Nous sommes presque prêts à lancer notre programme.

Lors du lancement du programme, Eclipse vous alerte des fichiers éventuellement non sauvegardés et vous propose de les sauvegarder. Vous pouvez toutefois à tout moment utiliser le menu **File → Save All** pour vous assurer que tous vos fichiers sont bien sauvegardés ou par le raccourci précédemment vu **Ctrl + Shift + S**. Notez également qu'un fichier non sauvegardé apparaîtra avec une astérisque dans l'éditeur, comme dans la capture Illustration 16: Fichier non sauvegardé.

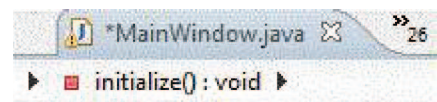


Illustration 16: Fichier non sauvegardé

Vérifiez d'abord que vous n'avez aucune erreur dans la vue «Problems» (capture Illustration 17: Liste des problèmes).

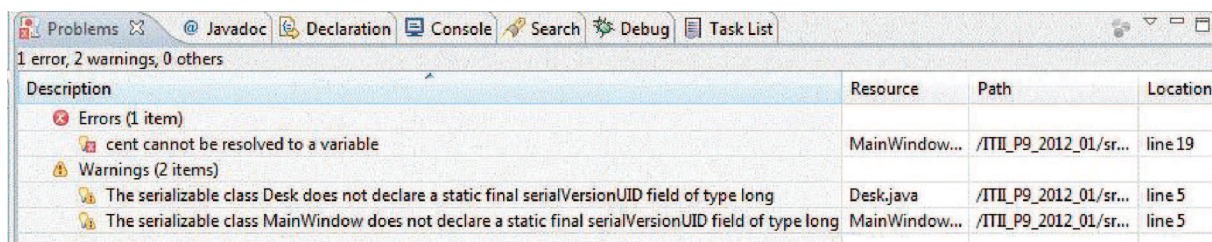


Illustration 17: Liste des problèmes

Si vous en avez, double-cliquer sur l'erreur vous mènera directement dessus. Vous pouvez ensuite obtenir des informations sur son origine en gardant le pointeur de votre souris au-dessus du symbole d'erreur dans la barre d'information de l'éditeur Eclipse (Capture Illustration 18: Message d'erreur depuis la barre d'information).

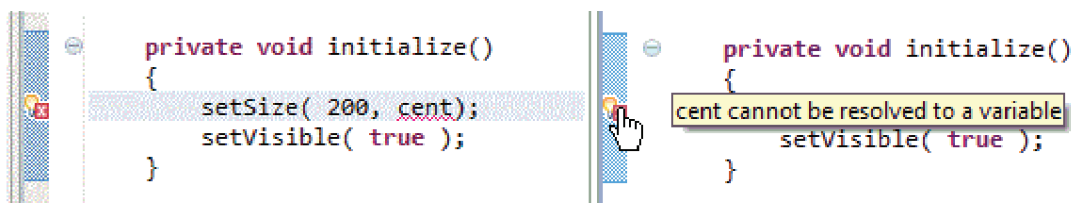


Illustration 18: Message d'erreur depuis la barre d'information

Nous allons maintenant lancer notre programme, qui pour l'instant ne fait rien d'autre que d'afficher une fenêtre.

Pour se faire, allez n'importe où dans votre classe contenant votre méthode '**main**' (Launcher) et faites un clique-droit (ou depuis la classe **MainWindow** depuis la vue **Package Explorer**). Dans le menu contextuel qui s'affiche, sélectionnez **Run As → Java Application** ; Cela exécutera votre

application et vous affichera votre interface aux dimensions spécifiées.

Rappelez-vous que désormais votre ‘programme’ est lancé. Cliquez sur la vue **Console** dans la liste des vues en bas de l’environnement Eclipse. Remarquez le gros carré Rouge qui vous permet à tout moment de forcer l’arrêt de votre programme ; C’est tout simplement le bouton de fin de programme. Lorsque votre programme est lancé, ce bouton devient actif (rouge), et lorsqu’il est arrêté, ou terminé, ce bouton se désactive (gris).

Fermez la fenêtre de votre programme, que constatez-vous dans votre vue Console ?

2- Expliquez

Arrêt du programme

Pour remédier à ce problème, nous allons ajouter une action par défaut lorsqu'on ferme la fenêtre au moment de l'initialisation de notre fenêtre.

```
setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

Relancez à nouveau l’application tout en gardant la vue **Console** sous les yeux. Vérifiez que votre précédente application avait bien été fermée (le gros carré rouge dans la vue **Console** ; illustration 18).



Illustration 19: Bouton d'état du programme

Fermez à nouveau l’application, que constatez-vous cette fois-ci ?

3- Expliquez

Coding guideline

Ces règles de codage sont là pour certifier que l’ensemble des différents codes écrits par la promotion respectent un certains standard.

Ainsi, l’ensemble des membres d’une classes seront privés à l’exception éventuelle des membres statiques et autres constantes.

L'utilisateur de ces membres se fera au travers d'accesseurs (getter / setter). Les *getters* seront en charge d'assurer que le membre retourné n'est jamais null.

Le constructeur vide devra être explicitement écrit (qu'il soit privé ou public).

La classe MainWindow sera un singleton puisqu'elle représente notre fenêtre principale.

L'application de gestion de Planning

A partir de maintenant, vous avez la liberté d'écrire le code comme bon vous semble en suivant les étapes décrites ci-après. Elles sont là pour vous guider et vous assurer que vous progressez dans la bonne direction avec un programme toujours stable et fonctionnel.

De plus, chaque chapitre détaille un peu plus ce que le cahier des charges ne fait que survoler. Ainsi, ces chapitres vous aideront dans le choix des bons composants graphiques et sur la logique à adopter pour arriver au résultat final.

Barre de menu

La barre de menu est une JMenuBar contenant des JMenu qui eux-mêmes peuvent contenir des JMenuItem s'ils possèdent des sous-menus.

Construisez l'arborescence des menus en respectant le cahier des charges et affichez le dans votre application.

Rappelez-vous que tous les composants graphiques devront être ajoutés au « `ContentPane` » de la `JFrame` (c'est à dire, de la **MainWindow**). Reportez-vous au cours si besoin.

Pour le moment nous ne coderons pas leurs actions. Ce sera vu ultérieurement lors de la création de la colonne des boutons.

Menu déroulant

Chaque élément du menu déroulant va afficher les mêmes informations – à savoir les tâches – sous une forme différent. Un menu déroulant est un `JComboBox`. Pour chaque élément de notre menu déroulant fera correspondra un `JPanel`.

L'astuce consistera ensuite à n'afficher que le bon `JPanel` correspondant à l'option choisie dans le menu déroulant parmi les 3 « `JPanel` ».

Nous aurons ainsi 3 `JPanels` nommés respectivement :

- `PanneauListe`
- `PanneauSemaine`
- `PanneauMois`

Pour le moment nos 3 `JPanels` seront vides. Pour distinguer chacun d'entre eux, vous pouvez par exemple modifier le « `background color` ». Le contenu des `JPanels` sera complété ultérieurement.

Nous désirons également que l'ensemble des éléments à afficher dans la liste soient issus d'une énumération (`Enum`) afin de s'assurer qu'aucun autre choix ne puisse être ajouté. De plus, cela facilitera l'ajout de nouveau choix à notre `JComboBox`.

Colonne de boutons

Sur la droite de l'application se trouve une colonne de `JButtons` représentant chacun une action.

Ces `JButtons` sont tous contenus dans un `JPanel` nommé **ButtonPanel** (Panneau Bouton).

Créez ce `JPanel` et ajoutez-y l'ensemble des `JButtons`.

Le cahier des charges indique l'action associée à chaque bouton. Reportez-vous y pour connaître le fonctionnement de chacun d'eux.

Fenêtre d'édition d'une tâche

Cette fenêtre sera la même pour les 3 actions suivantes : créer, éditer, dupliquer.

Le titre sera modifié en fonction de l'action en cours.

Les champs seront vierges si l'action est une création, ou préremplis pour une édition ou un dupliqué.

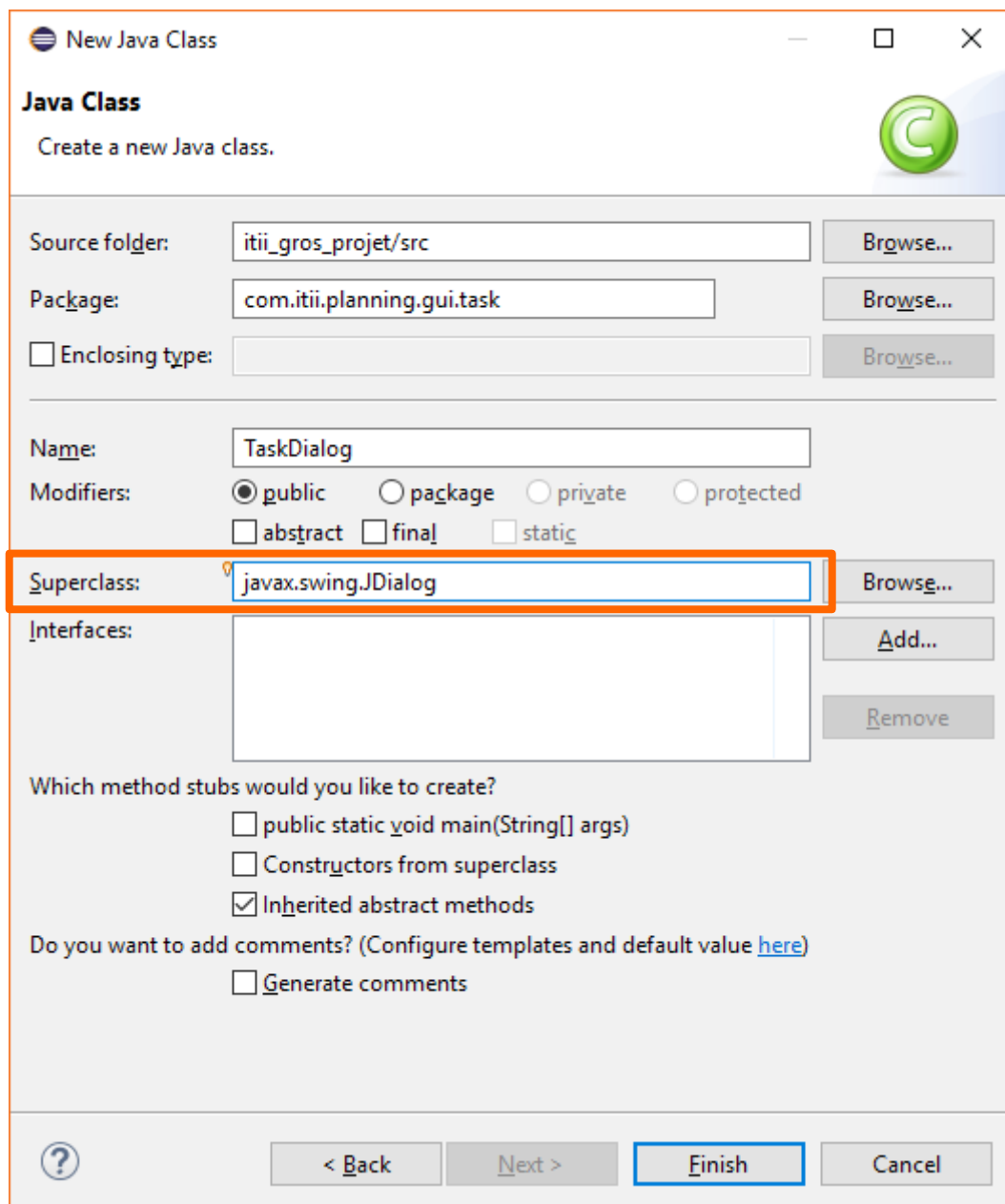
TP #2

Création de tâches

On souhaite que lorsque l'utilisateur clique sur le menu « Créer » ou sur le bouton « Créer » la fenêtre de création d'une tâche s'ouvre.

Pour cela, nous aurons besoin de cette nouvelle fenêtre qui héritera de JDialog.

Petite astuce sous Eclipse, vous pouvez créer une Classe en spécifiant dans la fenêtre de création la classe parent (ou Superclass).



Notez au passage que cette nouvelle classe (nommée « TaskDialog ») appartient à un nouveau

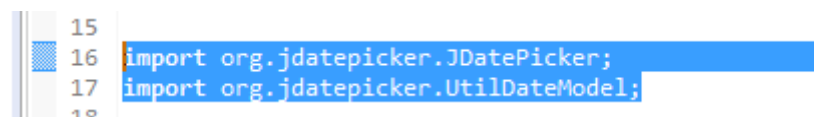
package « *com.itii.planning.gui.task* ». Rappelez-vous qu'il est important de bien structurer son projet.

Nous aurons besoin d'un composant permettant la sélection des dates dans le calendriers. Nous utiliserons (sauf si vous préférez en utiliser un autre) JDatePicker téléchargeable sur le site de l'éditeur: <https://jdatepicker.org>

Nous placerons cette nouvelle archive dans le répertoire C:\Users\##\eclipse-workspace\itii_gros_projet\lib que nous avons créé précédemment.

Nous allons ensuite rajouter cette archive au Build Path de Eclipse afin de rendre disponible à l'ensemble notre projet la bibliothèque contenue dans cette archive.

Pour utiliser cette nouvelle bibliothèque, il suffit de l'importer (*org.jdatepicker*) afin d'avoir accès à la nouvelle classe JDatePicker.



```
15
16 import org.jdatepicker.JDatePicker;
17 import org.jdatepicker.UtilDateModel;
18
```

Illustration 23: Import de Classes utiles

Le code ci-dessous vous montre un exemple d'implémentation de JDatePicker avec son modèle de données.

```
[...]
private JDatePicker calendar;
[...]
// Accesseur sur notre calendrier
public JDatePicker getCalendar()
{
    if (calendar == null)
    {
        UtilDateModel model = new UtilDateModel();
        Calendar cal = Calendar.getInstance();
        model.setDate(cal.get(Calendar.YEAR), cal.get(Calendar.MONTH),
            cal.get(Calendar.DAY_OF_WEEK));
        calendar = new JDatePicker(model, DateFormatUtil.DATE_FORMAT);
    }
    return calendar;
}
[...]
// Accesseur mettant à jour la date affichée
public void setDate(String date)
{
    getCalendar().getFormattedTextField().setText(date);
}
// Accesseur récupérant la date actuellement affichée
public String getDate()
{
    return getCalendar().getFormattedTextField().getText();
}
```

JTable

Pour vous aider dans la création de la vue Liste, il est intéressant d'utiliser une JTable. Ce composant permet d'afficher une table divisée en colonnes et en lignes avec une en-tête pour chaque colonne.

Pour l'instanciation de la table et l'ajout d'une en-tête :

```
DefaultTableModel tableModel = new DefaultTableModel(title, 0);  
JTable table = new JTable(tableModel);
```

Où *title* est un tableau contenant les titres à afficher pour chaque colonne. Title est un tableau de type : `String[]` dans lequel chaque entrée correspond à l'en-tête d'une colonne.

Le stockage et l'interaction avec les données de la table se fait au travers du « Data Model » de la JTable.

Pour accéder au Data Model de notre à notre table, il faut utiliser un « Table Model » (ici le `DefaultTableModel`). Dans la suite du programme, chaque fois que vous souhaitez accéder au Data Model, pensez à « caster » le modèle en `DefaultTableModel` comme suit :

```
( (DefaultTableModel) table.getModel() )
```

Ainsi, en manipulant l'objet `DefaultTableModel`, vous aurez accès à de nouvelles méthodes comme par exemple :

```
void javax.swing.table.DefaultTableModel.removeRow\(int row\)
```

```
void javax.swing.table.DefaultTableModel.addRow\(Object\[\] rowData\)
```

Qui permettent respectivement de supprimer ou d'ajouter une entrée dans la table.

Notez qu'ajouter une entrée consiste à fournis un tableau pour lequel chaque entrée correspond à un valeur d'une colonne. Ainsi, pour ajouter une tâche dans notre table qui posséderait les valeurs suivantes :

Il faudrait appeler la méthode ainsi :

```
((DefaultTableModel) planningList.getModel())  
    .addRow( new Object[] { "Terminer le TP",  
                           "2018/05/05 20:20",  
                           "Pour avoir une super note!" });
```