eric.blaudez@u-psud.fr

IUT 2015-2016  
TD / TP JAVA

Programmation avancée

|  |  |  |  |
| --- | --- | --- | --- |
| Nom | Prénom | e-mail | Groupe TD/TP |
| Bresson | Quentn | quentin.bresson@u-psud.fr | B2 |
| Ecollan | charles | [charles.ecollan@u-psud.fr](mailto:charles.ecollan@u-psud.fr) | B2 |
| Mouginot | Hubert | [Hubert.mouginot@u-psud.fr](mailto:Hubert.mouginot@u-psud.fr) | B2 |

Table des matières

[Evaluation 3](#_Toc430965353)

[Organisation des TD/TP 4](#_Toc430965354)

[Avant de commencer … 4](#_Toc430965355)

[A propos des TD/TP 4](#_Toc430965356)

[TD/TP 1 – Rappels 5](#_Toc430965357)

[Exercices 5](#_Toc430965358)

[Exercice 1 : Classe Abstraite, Interface & Héritage 5](#_Toc430965359)

[Exercice 2 : Collections 5](#_Toc430965360)

[Exercice 3 : GUI – Visualisation de Widgets sous forme de table : implémentation partielle d’un agenda 5](#_Toc430965361)

[Aide 6](#_Toc430965362)

[Rapport 6](#_Toc430965363)

[Description des travaux 6](#_Toc430965364)

[Difficultés rencontrées 6](#_Toc430965365)

[TD/TP 2 – Gestion des erreurs, tests & bonnes pratiques 7](#_Toc430965366)

[Exercices 7](#_Toc430965367)

[Exercice 1 : Tests unitaires et *debugger* 7](#_Toc430965368)

[Exercice 2 : Internationalisation 7](#_Toc430965369)

[Exercice 3 : Gestion des erreurs (Exceptions) 7](#_Toc430965370)

[Exercice 4 : GUI – Changer la langue d’une application 7](#_Toc430965371)

[Exercice 5 : Créer sa bibliothèque & construire un livrable 7](#_Toc430965372)

[Rapport 7](#_Toc430965373)

[Description des travaux 7](#_Toc430965374)

[Difficultés rencontrées 7](#_Toc430965375)

[TD/TP 3 – Généricité, Collections & *Design patterns* 8](#_Toc430965376)

[Exercices 8](#_Toc430965377)

[Exercice 0 8](#_Toc430965378)

[Exercice 1 : Généricité – Création d’un lecteur d’arguments 8](#_Toc430965379)

[Exercice 2 : Collections - créer une collection d’***ExamEvent*** 8](#_Toc430965380)

[Exercice 3 : Design pattern 8](#_Toc430965381)

[Exercice 4 : GUI – Menu 8](#_Toc430965382)

[Aide 9](#_Toc430965383)

[Rapport 9](#_Toc430965384)

[Description des travaux (+réponses aux questions) 9](#_Toc430965385)

[Difficultés rencontrées 9](#_Toc430965386)

[TD/TP 4 – Les entrées / sorties 10](#_Toc430965387)

[Exercices 10](#_Toc430965388)

[Exercice 1 : Lire & écrire un fichier XML 10](#_Toc430965389)

[Exercice 2: Créer et lire un fichier de configuration pour une application 10](#_Toc430965390)

[Exercice 3 : Sauvegarder et charger l’état d’une application 10](#_Toc430965391)

[Exercice 4 : GUI 10](#_Toc430965392)

[Exercice 5 - facultatif: Base de données 10](#_Toc430965393)

[Aide 10](#_Toc430965394)

[Rapport 10](#_Toc430965395)

[Description des travaux 10](#_Toc430965396)

[Difficultés rencontrées 10](#_Toc430965397)

[TD/TP 5 – *Threads* & Calcul distribué 11](#_Toc430965398)

[Exercices 11](#_Toc430965399)

[Exercice 1 : Sauvegarde en tâche de fond 11](#_Toc430965400)

[Exercice 2 : ***ShutdownHook*** 11](#_Toc430965401)

[Exercice 2 : Lancer des calculs distribués 11](#_Toc430965402)

[Exercice 3 : GUI – configuration 11](#_Toc430965403)

[Rapport 11](#_Toc430965404)

[Description des travaux 11](#_Toc430965405)

[Difficultés rencontrées 11](#_Toc430965406)

[TD/TP 6 – Les applications en réseau 12](#_Toc430965407)

[Exercices 12](#_Toc430965408)

[Exercice 1 : Créer une application de *Chat* 12](#_Toc430965409)

[Exercice 2 : Créer une *Shoutbox* avec une *servlet (ou une page JSP)* 12](#_Toc430965410)

[Rapport 12](#_Toc430965411)

[Description des travaux 12](#_Toc430965412)

[Difficultés rencontrées 12](#_Toc430965413)

# Evaluation

|  |  |  |
| --- | --- | --- |
| td /tp | Note | Commentaire |
| 0 - Github | /2 |  |
| 1 | /3 |  |
| 2 | /3 |  |
| 3 | /3 |  |
| 4 | /3 (+1) |  |
| 5 | /3 |  |
| 6 | /3 |  |

# Organisation des TD/TP

## Avant de commencer …

* Ces TD/TP sont accompagnés des codes sources qu’il s’agit de compléter.
* Il est **obligatoire** de créer un compte **GitHub** (***https://github.com***) par groupe de travail (normalement un binôme) **avant** le premier TD/TP (vous fournirez les adresses à vos chargés de TD). Chaque semaine, vous y déposerez vos travaux. La documentation de **GitHub** est disponible sur ***https://help.github.com/***
* Une copie de ce document se trouve dans le dossier « ***train*** », il devra être complété (nom, prénom, email et groupe - en première page) et *versionné* au même titre que vos travaux.
* La hiérarchie de dossiers (*versionnées*) devra être la suivante (le fichier zip avec le premier TP respecte cette hiérarchie, il suffira de *versionner* son contenu)

***Dossiers***

***Workspace Eclipse***

***Fichiers***

## A propos des TD/TP

* A partir du TP 2, tous les codes fournis devront être **commentés** en ***JavaDoc*** et les classes que vous jugez critiques devront faire l’objet de **tests unitaires** (il faudra effectuer ce travail sur le TP1 également). Une note comprise entre 0,25 et 0,75 par TP sera donnée pour ce travail.
* Le code source à compléter est une base de travail, vous pouvez ajouter ou supprimer des objets.
* A l’issu de chaque TD/TP, vous **devez** compléter la partie « ***rapport*** » de ce document afin d’expliquer votre démarche et les difficultés rencontrées. Vous pouvez ajouter des diagrammes de classes et d’états.
* Tous les TP sont basés sur le même code : chaque TP est donc une amélioration du TP précédent. Sur votre compte ***GitHub***, il sera inutile de faire un dossier par TP, vous compléterez l’existant : suivez la hiérarchie de dossier proposée.
* La plupart des exercices seront utiles pour le projet.

# TD/TP 1 – Rappels

## Exercices

### Exercice 1 : Classe Abstraite, Interface & Héritage

Pour l’exercice suivant les classes se situent dans les *packages* ***edu.iut.app*** et ***edu.iut.gui.listeners***

* Compléter l’interface ***IApplicationLogListener*** ayant une méthode ***void newMessage(String level, String message)***
* Compléter une interface ***IApplicationLog*** ayant les méthodes suivantes
* ***void setMessage(String message)***
* ***String getMessage()***
* ***void addListener(IApplicationLogListener listener)***
* ***IApplicationLogListener[] getpplicationLogListeners()***
* Compléter la classe ***AbstractApplicationLog*** implémentant l’interface ***IApplicationLog***
  + La méthode ***setMessage*** sera abstraite
  + Cette classe devra contenir un tableau de ***IApplicationLogListener*** (sous forme de collection) avec une méthode d’ajout et une méthode permettant d’obtenir le tableau/
* Compléter les classes ***ApplicationErrorLog, ApplicationWarningsLog, ApplicationInfoLog*** dérivant de ***AbstractApplicationLog*** 
  + Ces classes devront contenir un tableau de ***IApplicationLogListener*** (sous forme de collection)
  + La méthode ***setMessage*** doit appeler la méthode ***newMessage*** de chaque élément du tableau de ***IApplicationLogListener*** et conserver le message
* Compléter la boîte de dialogue abstraite ***AbstractApplicationMessageDialog*** implémentant l’interface ***IApplicationLogListener***
  + ajouter une méthode abstraite ***void showMessage( )***, cette méthode doit être accessible par les classe filles seulement.
  + la méthode ***void newMessage(String message)*** doit appeler la méthode ***showMessage( )***
* Compléter les 3 boites de dialogue dérivant de ***AbstractApplicationMessageDialog***:
  + ***ApplicationErrorMessageDialog ,*** affichant un message d’erreur
  + ***ApplicationWarningMessageDialog***, affichant un avertissement
  + ***ApplicationInfoMessageDialog***, affichant un message

### Exercice 2 : Collections

Créer une table de logs ***ApplicationLogs***

* La classe ***ApplicationLogs*** doit dériver d’une Collection représentant un tableau de ***IApplicationLog***
* Implémenter les fonctions suivantes permettant de catégoriser les erreurs
  + ***ArrayList<IApplicationLog> getErrors(),***
  + ***ArrayList<IApplicationLog> getWarnings(),***
  + ***ArrayList<IApplicationLog> getInfos()***

### Exercice 3 : GUI – Visualisation de Widgets sous forme de table : implémentation partielle d’un agenda

* Implémenter 3 vues (Mois / Semaine / Jour) : classes ***EventPanel, MonthPanel, WeekPanel, DayPanel***
* Trouver un *Layout* permettant de superposer ces trois vues : classe ***SchedulerFrame***
* Créer une ***Factory*** (classe ***AgendaPanelFactory***)permettant de gérer les trois classes ***MonthPanel, WeekPanel, DayPanel***
* Créer une Panel permettant de gérer les vues ***MonthPanel, WeekPanel, DayPanel :*** en cliquant sur un bouton ‘***next***’ nous devons pouvoir voyager entre les vues

## Aide

* Operateur ***instanceof***
* ***JOptionPane***: <https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-menus-et-boites-de-dialogue>

## Rapport

### Description des travaux

Pour ce TP il nous a été demandé de compléter diverses classes de manière à rendre l’application opérationnelle (beaucoup d’erreur était présente au début, surtout des vides, que l’on a remplis).

Ce TP nous as demandé de nombreuses connaissances acquises l’année passée (l’héritage, les interfaces graphique, les interfaces …) et nous a permis de consolider nos compétences.

Ainsi dans l’exercice un nous avons complété diverses classes du package edu.iut.app afin de rendre utilisable les listeners qui nous serviront plus tard à gérer les erreurs. L’exercice deux était la suite logique de l’exercice un, il nous demandait de compléter des fonctions permettant d’obtenir de trier les messages suivant leur type (info, warning ou error).

L’exercice trois nous a quant à lui permis d’implémenter une interface graphique représentant un agenda avec trois vues différentes : les jours, les semaines et les mois.

### Difficultés rencontrées

Nous avons rencontré quelques difficultés au commencement du TP, en effet la taille du projet déjà en place nous a freinés dans notre progression. Nous avons dû réfléchir longuement à la manière dont le programme de base était construit, quel classe interagissait avec une autre, etc

La principale difficulté est venue de l’exercice trois, dans lequel il nous était demandé d’implémenter trois vues, de créer un layout gérant ces trois vues, une classe *factory*, et un panel gérant les vues. En effet nous avons pu constater que ces classes étaient déjà pour la plupart complète dans le programme de base, ce qui nous a intrigués. Nous avons donc complété l’existant comme nous le pouvions, en rajoutant par exemple dans la classe *MonthPanel* un code permettant de trouver le premier jour du mois actuel et d’élaborer la vue à partir de celui-ci.

# TD/TP 2 – Gestion des erreurs, tests & bonnes pratiques

## Exercices

### Exercice 1 : Tests unitaires et *debugger*

* Pour chaque classe ‘non graphique’ du TP, créer un test unitaire.
* A l’aide du débugger, trouver l’(es) erreur(s) introduite(s) dans le code du TP2.

### Exercice 2 : Internationalisation

Créer un ***Singleton*** session, permettant de gérer l’internationalisation (français / anglais) ***edu.iut.app.ApplicationSession***

### Exercice 3 : Gestion des erreurs (Exceptions)

* Créer vos propres classes d’exceptions : créer une classe ***IUTException*** de laquelle dériveront toutes vos futures classes d’exception. ***IUTException*** devra pouvoir ‘logger’ les erreurs : en vous appuyant sur la classe de Session pour initialiser le ***logger***.
* Dans le TP1, nous avons créé les classes ***ApplicationErrorLog, ApplicationWarningsLog, ApplicationInfoLog*** permettant une gestion avancée des erreurs utilisant des ***listeners*** (permettant d’afficher des messages à l’écran). Il faudrait que ces messages soient aussi ‘***loggés***’ : modifier ces classes afin d’utiliser un ***logger*** initialiser dans la session.

### Exercice 4 : GUI – Changer la langue d’une application

En vous appuyant sur le code d’internationalisation de la session, faite en sorte que les éléments affichés (dans l’interface) ou dans les erreurs, soient dépendant de ***Locale*** (autrement dit, l’application devra pouvoir afficher de l’anglais et du français sans avoir été recompilée).

### Exercice 5 : Créer sa bibliothèque & construire un livrable

A l’aide d’***Eclipse***, créer une application (un ‘***.jar***’) livrable (à vos chargés de TD par exemple)

***Ce livrable devra désormais être disponible tout le temps (pour chaque TD/TP ou à chaque évolution majeur de vos codes, il faudra le reconstruire). Il faut donc le ‘versionner’ dans le dossier train. Les scripts ‘run.sh’ et ‘run.bat’ permettent d’exécuter votre application.***

## Rapport

### Description des travaux

Ce TP nous a initiés à JUnit et aux tests unitaires. Il fut en effet nécessaire d’élaborer des classes tests afin de tester des sections critiques du programme (implémentées dans les TP suivants).

Ce TP a également été l’occasion de créer un singleton (classe ApplicationSession) représentant une session ouverte par l’utilisateur. Nous nous sommes ainsi initiés à l’internationalisation en découvrant les classes ResourceBundle et Logger, qui nous ont permis de modifier la langue de notre programme en fonction du lieu de résidence de l’utilisateur.

Nous avons également découvert comment créer un exécutable (fichier .jar) de notre programme à partir d’Eclipse.

### Difficultés rencontrées

Ce TP nous a initiés à JUnit et aux tests unitaires. Il fut en effet nécessaire d’élaborer des classes tests afin de tester des sections critiques du programme (implémentées dans les TP suivants).

Ce TP a également été l’occasion de créer un singleton (classe ApplicationSession) représentant une session ouverte par l’utilisateur. Nous nous sommes ainsi initiés à l’internationalisation en découvrant les classes ResourceBundle et Logger, qui nous ont permis de modifier la langue de notre programme en fonction du lieu de résidence de l’utilisateur.

Nous avons également découvert comment créer un exécutable (fichier .jar) de notre programme à partir d’Eclipse.

# TD/TP 3 – Généricité, Collections & *Design patterns*

## Exercices

### Exercice 0

Pourquoi, dans le TP1, a-t-on créé un ***Singleton*** (classe ***ApplicationSession***) ?

### Exercice 1 : Généricité – Création d’un lecteur d’arguments

* Créer une classe générique **Option,** Les options peuvent être des booléens, des entiers, des réels, des chaînes de caractères ou encore des fichiers. Une option est caractérisée par une clé, une valeur et la description de l’option (Note : pensez à l’internationalisation). Ici vous devrez travailler sur la classe ***CommandLineOption.***
* Créer une classe ***CommandLineParser***
* Modifier le programme principal pour utiliser votre ***parser*** d’options

### Exercice 2 : Collections - créer une collection d’***ExamEvent***

* Créer les classes :
  + Person,
  + Classroom,
  + Document
  + … toutes celles qui vous semblent nécessaires
* Créer la classe ***ExamEvent*** (cette classe doit répondre à la problématique soumise dans le projet … à vous d’en faire le ***design***). Elle doit contenir a minima un objet ***Person*** (celle passant l’examen), une collection de ***Person*** (le jury), un objet ***Classroom*** et naturellement toutes les informations ‘normales’ pour évènement.
* Créer la class ***Agenda*** dérivant d’une collection d’***ExamEvent***

### Exercice 3 : Design pattern

Créer un Filtre (voir aide) sur la collection ***ExamEvents***

* Critère pour les dates
* Critère pour les personnes
* Critère pour les salles
* Imaginer et implémenter d’autres critères (voir le projet)

### Exercice 4 : GUI – Menu

* Ajouter un menu à votre application

Implémenter les fonctionnalités du ‘menu’ ***Edit*** : les ***menuitems*** de ‘***View***’ doivent repoduire le comportement du bouton ‘***Next***’ de l’interface. Pour les autres menus, afficher un message (***DialogBox***) d’information prévenant l’utilisateur que la fonctionnalité n’est pas encore implémentée.

* Remplacer le bouton ‘***Next***’ par
  + Un ***Spinner*** contenant les années de 2010 à l’année 2020 (positionné sur l’année courante)
  + Une ***ComboBox*** contenant les mois de l’année (positionné sur le mois courant)
  + Une ***ComboBox*** contenant les jours de la semaine (positionné sur le jour courant)

(Pensez à l’internationalisation)

* Créer les widgets pour l’affichage d’un ‘***ExamEvent***’, en lien avec une Collection d’***ExamEvent*** (Exercice 2). Ces widgets devront permettre l’ajout/la suppression et la modification d’un ***ExamEvent*.**

## Aide

**Filter Pattern** : <http://www.tutorialspoint.com/design_pattern/filter_pattern.htm>

**Menu :** <https://docs.oracle.com/javase/tutorial/uiswing/components/menu.html>

**Spinner :** <https://docs.oracle.com/javase/tutorial/uiswing/components/spinner.html>

## Rapport

### Description des travaux (+réponses aux questions)

Nous avons utilisé un singleton pour la session afin de garantir le fait qu’une seule et unique instance de la classe correspondant à une session sera créée. Cela permet de correctement implémenter un système de logging centralisé, ce qui est le but recherché ici.

Nous avons commencé par utiliser la généricité pour traiter les options lancées avec l’application (par exemple : mon projet –langue fr ; on utilise la généricité pour effectuer les traitements correspondant à l’option –langue fr). Cet exercice nous a donc initiés au concept de la généricité dans un langage orienté objet.

Ensuite nous avons dû implémenter les classes permettant de gérer l’organisation des soutenances (ExamEvent ou Person par exemple). Après l’implémentation des classes nous avons utilisé le design pattern *filter pattern* afin d’effectuer des tris sur les personnes qui par exemple, n’ont pas encore de soutenance, ou rechercher des plages horaires ou des salles encore disponibles.

Pour finir nous avons amélioré l’interface graphique de notre application en y ajoutant un menu *menubar* et en utilisant des spinners ou encore des actionlisteners au sein de celui-ci.

### Difficultés rencontrées

La première difficulté s’est présenté lors de l’implémentation des classes gérant les options, notamment avec la généricité. On était un peu perdu au départ, on ne savait pas quelle voie prendre, on en a choisi une sans être sûr de si c’était la bonne.

La principale difficulté s’est posé lors de l’implémentation du *filter pattern.* Les classes de critère ont été implémenté en utilisant la généricité (en effet le pattern de base a été modifié afin de mieux correspondre au concept à implémenter). Le choix des critères sur lesquels faire divers tris a été long, mais pas spécialement difficile à implémenter une fois choisi. Le problème s’est posé lors de l’implémentation de la classe testant l’ensemble des critères, qui utilise de nombreux concepts.

La dernière difficulté se posa sur l’implémentation du menu. Il nous a fallu mettre à jour nos connaissances des interfaces graphique (JFrame, JPanel, ActionListeners …) pour surmonter ce problème. En effet n’ayant pas de talent particulier pour le graphisme, arriver à un résultat satisfaisant fut compliqué.

# TD/TP 4 – Les entrées / sorties

## Exercices

### Exercice 1 : Lire & écrire un fichier XML

* Sauvegarder le contenu de l’application (agenda) dans un fichier XML.
* Utiliser ***CommanLineParser*** (TP3) avec l’option ***–project=<FILE>*** pour pouvoir charger un contenu d’application préalablement sauvegarder.

### Exercice 2: Créer et lire un fichier de configuration pour une application

* La configuration doit contenir le chemin du fichier de log et le chemin d’un dossier temporaire (il faudra donc ajouter dans la session la possibilité de ***loggé*** dans un fichier et les accesseurs fournissant le chemin vers ce dossier temporaire)
* Utiliser ***CommanLineParser*** (TP3) avec l’option ***–config=<FILE>*** pour pouvoir charger une configuration

### Exercice 3 : Sauvegarder et charger l’état d’une application

Sérialiser les objets contenant de l’information dans le dossier temporaire (initialiser avec la configuration)

### Exercice 4 : GUI

* Lire et visualiser une aide HTML (vous pouvez activer le ***menuitem*** ‘help’ et ‘about’)
* Activer les ***menuitems*** ‘save’ et ‘load’
* Ajouter une interface permettant d’ajouter/modifier une personne (ces informations seront sauvegardées à part dans un fichier XML)

### Exercice 5 - facultatif : Base de données

Vous pouvez ajouter une fonctionnalité offrant la possibilité d’effectuer une sauvegarde dans une base de données.

## Aide

* Afficher de l’HTML : <http://alvinalexander.com/blog/post/jfc-swing/how-create-simple-swing-html-viewer-browser-java>
* Base de données : <http://www.tutorialspoint.com/sqlite/sqlite_java.htm>

## Rapport

### Description des travaux

### Difficultés rencontrées

# TD/TP 5 – *Threads* & Calcul distribué

## Exercices

### Exercice 1 : Sauvegarde en tâche de fond

En utilisant un Thread et le système de sauvegarde du TD/TP 4 (exercice 1), sauvegarder le projet toute les minutes.

### Exercice 2 : ***ShutdownHook***

En utilisant le code implémenté dans le TD/TP 4 (exercice 3) écrire une ***ShutdownHook*** pour faire un système de reprise sur erreur en cas de crash.

### Exercice 2 : Lancer des calculs distribués

Dans le TP2 nous avons créé un Filtre, il s’agit ‘de l’externaliser’ afin que les recherches selon un critère (qui peut être complexe) soient effectuées sur une autre machine.

Cette fonctionnalité ne doit en rien casser l’existant : l’utilisateur devra pouvoir utiliser des calculs externalisés ou bien ses propres ressources.

Ajouter dans le fichier de configuration l’adresse du server RMI, avec un booléen permettant d’activer (ou de désactiver) la fonctionnalité.

### Exercice 3 : GUI – configuration

Créer une ‘***DialogBox***’ *Settings* permettant de configurer entre autre l’***autosave*** (est-ce que l’***autosave*** est activé ? à quelle fréquence ?).

* Cette ‘***DialogBox***’ devra être accessible depuis le menu (***edit->settings***)
* La configuration devra être sauvegardée dans un fichier de configuration
* Les ***settings*** ne concernent pas uniquement la sauvegarde : ajouter d’autre settings (comme le chemin pour le fichier log, l’adresse du server RMI ou encore la langue par exemple)

## Rapport

### Description des travaux

### Difficultés rencontrées

# TD/TP 6 – Les applications en réseau

## Exercices

### Exercice 1 : Créer une application de *Chat*

* Prévoir un mode ‘***public’*** et ‘***private***’
* Ajouter dans le fichier de configuration l’adresse et le port de votre server

### Exercice 2 : Créer une *Shoutbox* avec une *servlet (ou une page JSP)*

* Cette ***shoutbox***  devra se connecter au server pour afficher les commentaires ‘public’ uniquement
* Il sera aussi possible de poster un commentaire

## Aide

***Shoutbox*** : <https://fr.wikipedia.org/wiki/Shoutbox>

## Rapport

### Description des travaux

### Difficultés rencontrées