LABORATOIRE 15 | *Informatique 1*
PROGRAMMATION ORIENTEE OBJETS

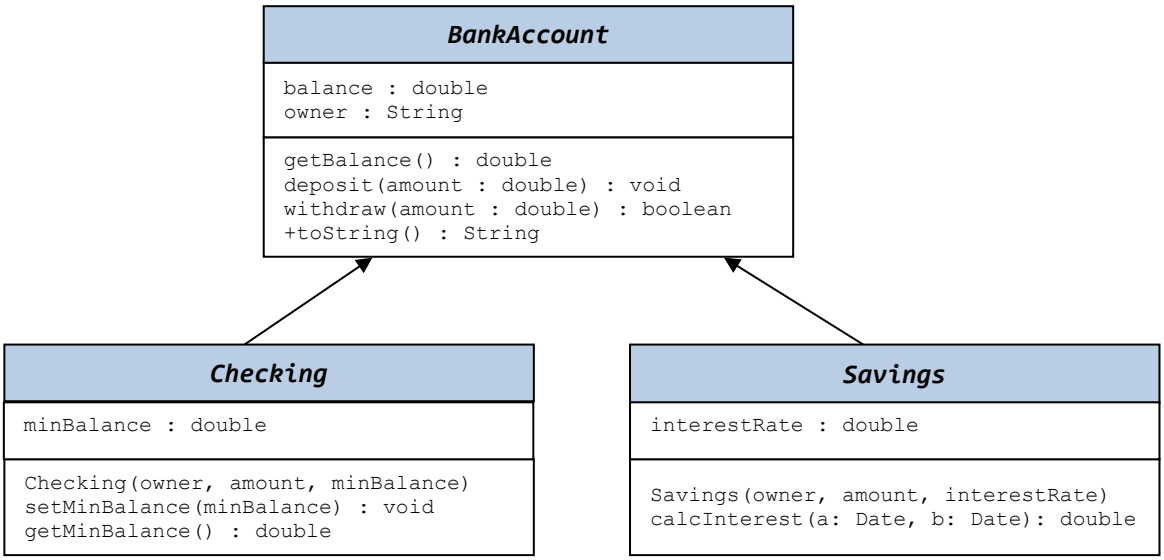
Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

But du laboratoire (10 périodes)

1. Ce laboratoire est le dernier laboratoire avant de passer au projet. Le labo est divisé en trois parties principales : la première partie concerne l'application de l'héritage dans une application bancaire et le reste du labo est articulé autour des applications graphiques. Notamment, vous allez ainsi pouvoir mettre en pratique toutes vos connaissances de Java pour réaliser une application complète "professionnelle".
2. La durée estimée pour réaliser ce laboratoire est de **dix périodes**. Si le temps imparti ne devait pas suffire, vous êtes invités à terminer le travail en dehors des heures de cours.
3. Vous pouvez trouver cette donnée sous forme électronique sur http://inf1.begincoding.net.
4. Pour ce laboratoire, vous devrez présenter votre programme et vous ***serez interrogés par oral sur le code***. Vous trouverez plus de détails à la fin de ce document. Veuillez ***bien lire*** cette partie !

Partie 1 - Héritage, compte en banque et intérêt

Dans cette partie, nous vous proposons un exercice afin que vous puissiez pratiquer en labo les notions vues sur l'héritage. Vous devez ainsi réaliser une application bancaire très simple dans laquelle la hiérarchie de classes suivante est utilisée (vous devez choisir les modificateurs, sauf pour la méthode `toString` qui est complète):



Dans cette banque, il existe deux types de compte : les comptes courants (*checking account*) et les comptes d'épargne (*savings account*). Les caractéristiques de ces comptes sont les suivantes :

1. Les comptes courants ne donnent pas droit à un intérêt mais il est ***toujours*** possible (même à la création du compte) d'avoir un solde négatif sur le compte jusqu'à une certaine limite (qui est configurable par la personne s'occupant du compte à la banque). Il s'agit du `minBalance` dans la représentation UML. Notez que ce chiffre qui peut être modifié par la méthode `setMinBalance` doit ***toujours*** être négatif (ou égal à 0). Par exemple, si `minBalance` à une valeur de -10000 (notez le signe moins), il est possible d'avoir un solde négatif de dix mille francs. Si la personne qui gère le compte essaie de retirer de l'argent et que cela n'est pas possible, un message d'erreur doit être affiché et le compte ne doit pas être modifié.

2. Les comptes d'épargne donnent droit à un intérêt mais il est impossible d'avoir un solde négatif. Un compte épargne possède une méthode permettant de calculer l'intérêt du compte entre deux dates. Notez que pour calculer le nombre de jours entre deux dates, vous pouvez utiliser la classe `DateUtils` qui se trouve dans le paquetage `hevs.utils`.

Tâche 1

1. Créez un nouveau projet *Eclipse* nommé *lab15*.
2. Ajoutez un paquetage que vous nommerez `bank`.

3. Implémentez la classe `BankAccount` en prenant en compte le fait qu'il doit être ***impossible d'instancier*** cette classe directement.
4. Implémentez les classes `Checking` et `Savings` correspondant aux descriptions UML faites ci-dessus. Ces descriptions sont volontairement incomplètes.

Tâche 2

Il est important qu'un programme bancaire ne comporte pas d'erreurs. Ainsi, si la personne qui gère le compte essaie de retirer de l'argent et que cela n'est pas possible, un message d'erreur doit être affiché et le compte ne doit pas être modifié. Veuillez utiliser le code suivant pour vos messages d'erreur (sans retour à la ligne ou autres tabulations) :

```
System.out.println("Problem : cannot withdraw that amount");
```

ce qui donne sur la console le message suivant

```
Problem : cannot withdraw that amount
```

Ce comportement (afficher un message commençant par `"Problem : "` + ne pas modifier le compte en cas de problèmes) doit être le même quel que soit le type d'erreur que l'on puisse rencontrer.

1. Ajoutez tous les tests possibles pour éviter tous les problèmes pouvant arriver (intérêt négatif...).

Pour vérifier que vous n'avez pas fait d'erreurs, nous allons maintenant tester que votre code fonctionne correctement. L'avantage d'avoir utilisé une description UML est que l'interface de votre classe, c'est-à-dire les méthodes et les attributs qu'elle possède, sont standardisés entre vous tous. Il est dès lors possible d'écrire une classe qui va utiliser les classes que vous venez d'écrire et les tester.

Afin d'augmenter la sécurité de nos tests, nous avons utilisé pour écrire la classe de test le mécanisme des *assertions*. Des explications seront données en classe mais, en un mot, il s'agit d'un nouveau mot-clé qui permet de dire qu'une expression Java doit valoir `true` sinon le programme s'arrête. Pour que les assertions fonctionnent, il faut rajouter `-ea` comme paramètre de la machine virtuelle (menu *Run Configuration* d'*Eclipse*).

1. Pour tester votre classe, nous vous fournissons une classe `BankController` qui contient du code qui va vérifier que le code que vous avez écrit fonctionne comme prévu.
2. Pour lancez les tests, utilisez le `main` se trouvant dans la classe `BankController`.
3. Dans un cas idéalisé, tous les problèmes doivent être détectés et aucune erreur ne doit être possible avec votre programme (il s'agit quand même de gérer de l'argent !). Toutefois, sachez que cela ne sera probablement pas le cas, ce qui est **tout à fait normal**.

   En effet, même les programmeurs-ses chevronné-e-s laissent des bugs derrière eux-elle. Il est donc important de prévoir des tests comme celui-ci pour pouvoir vérifier que vos programmes fonctionnent comme vous l'avez prévu[1].

Partie 2 - Premiers pas avec Swing

Vous allez maintenant pouvoir développer votre première interface graphique utilisant *Swing*, le gestionnaire graphique de base offert par Java. N'oubliez pas de consulter les vidéos correspondantes sur le site web du cours !

Tâche 1

Dans cette première tâche, vous devez créer une fenêtre qui affichera un fond vert avec un bouton sur le devant.

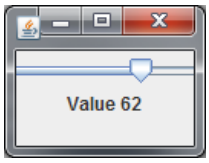
1. Créez une classe nommée `GUI1`. Pour cela, inspirez-vous des exemples donnés au cours. Faites attention de faire hériter votre classe de la classe `JFrame`.

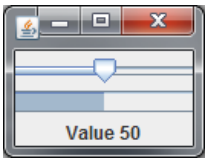
[1] La question du test des programmes est un sujet à part entière que nous n'avons malheureusement pas le temps d'aborder. Les personnes intéressées peuvent trouver quelques informations dans la bibliographie présente à la fin de ce document.

2. Créer un listener nommé `BasicListener` qui permettra de réagir aux pressions sur le bouton en changeant la couleur du fond de la fenêtre. Une pression sur le bouton donnera une fois un fond bleu, une fois un fond vert.

Tâche 2

Dans cette deuxième tâche vous allez utiliser deux composants que nous n'avons pas vu ensemble, le `JSlider` et la `JProgressBar`. Dans un premier temps, le but est que lorsque l'on déplace le slider, le texte affiché dans un label change comme dans l'exemple ci-dessous (figure de gauche) :



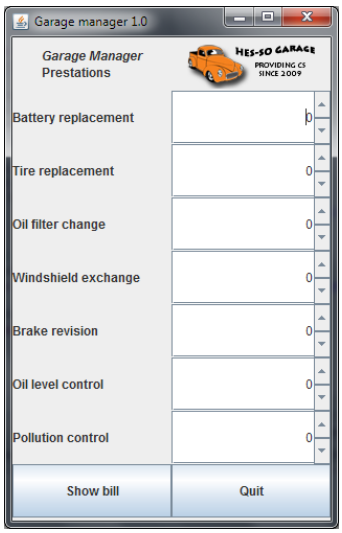


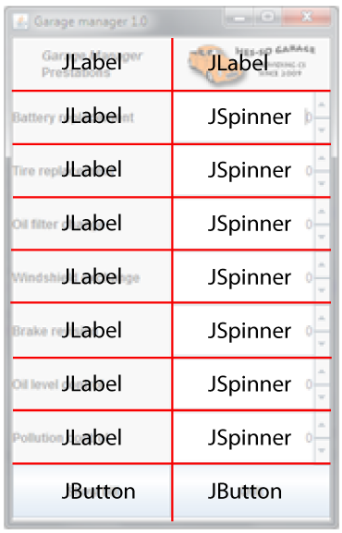
1. Créez une classe `GUI2`. Ajoutez-y ce qu'il faut pour obtenir ce qu'il y a sur la figure de gauche ci-dessus. Attention, les évènements générés par le slider ne sont pas des `ActionEvent` (voir slides cours) !
2. Dans un deuxième temps, ajoutez une `JProgressBar` pour obtenir le comportement démontré sur la figure de droite.

Partie 3 - Générateur graphique de factures

Dans cette partie, vous allez mettre en application vos nouvelles connaissances sur les interfaces graphiques en améliorant un programme qui était relativement inutilisable par un-e béotien-ne. En effet, nous avons vu durant le laboratoire 13 sur les streams une application de garage pour générer des factures. Toutefois, générer différentes factures demandait de changer le contenu d'un tableau Java. Le but de cette partie est de fournir une interface graphique agréable afin que des utilisateurs puissent créer des factures différentes sans avoir à entrer dans le code.

Nous vous proposons de réaliser la GUI suivante (figure de gauche):





GridLayout

Cette GUI utilise un `GridLayout` qui permet de mettre en place les composants en suivant une grille comportant n lignes et m colonnes (on spécifie le nombre de lignes et de colonnes dans le constructeur). Dans notre exemple, nous avons ainsi 9 lignes et 2 colonnes.

Fonctionnement de l'application

L'application doit permettre d'entrer un certain nombre de prestations puis de générer, lorsque l'on presse le bouton "*Show Bill*", il faut afficher la facture sous forme HTML. La génération du code HTML à proprement parler est déjà faite par la classe `GarageManager` (méthode `generateHTMLBill`)

Quelques remarques :

1. Le texte des prestations ainsi que leur nombre doivent être récupéré depuis la classe `GarageManager`. Ainsi, Le nombre de lignes de cette interface dépend du nombre de prestations offertes par la classe `GarageManager`.
2. La première ligne de la GUI comporte en fait deux `JLabel`. En effet, un `JLabel` peut contenir du texte (dans notre cas sur deux lignes) mais également des images comme c'est le cas du logo du garage que l'on voit.

Utilisation de l'HTML dans les JComponent (composants)

Il est possible d'utiliser de l'HTML dans les composants Java en commençant le string utilisé par <html> et de le terminer par </html>. Notez au passage qu'il faut "ouvrir" les tags et les refermer (à l'aide du caractère '/'), ceci afin d'indiquer la fin de la validité du tag.

Voici une petite liste des tags existants et que vous pouvez utiliser directement partout où de l'HTML est utilisable :

Tag	Name	Example	Output
****	Bold	Hello	**Hello**
** **	Center	Hello	Hello
****	Emphasis	This is an ****Example****	This is an *Example*
** **	Line break	Hello World	Hello World

Vous trouverez sur Internet des listes plus complètes, par exemple pour réaliser des listes numérotées etc… (par exemple http://www.web-source.net/html_codes_chart.htm).

Model-view-controller

Notre application sépare complètement les données de leur affichage dans la GUI. En effet, `GarageManager` s'occupe de générer les factures sous forme de fichier HTML et vous devez pour cela lui fournir un tableau Java. La partie traitement est ainsi complètement dévolue à la classe GarageManager alors que les interactions se passeront dans une autre classe que nous nommerons `ManagerGUI`. Cette séparation stricte est connue en informatique sous le nom de modèle *MVC* (model-view-controller) et constitue une très bonne manière de gérer des programmes avec GUI. Vous trouverez des références sur ce sujet dans la bibliographie.

Tâche 1

1. Observez-bien la classe `GarageManager` afin de comprendre les méthodes qu'elle vous offre et que vous pouvez utiliser !
2. Créez une nouvelle classe, nommée `ManagerGui` qui contiendra le `main` et l'interface graphique.
3. Créez les différents composants nécessaires pour votre interface en vous inspirant librement de la figure ci-dessus. Libre à vous de choisir quelque chose de différent mais globalement les mêmes fonctionnalités doivent être disponibles.

   Pour vous aider, il existe des composants déjà tout fait, nommés `JSpinner`, que vous pouvez utiliser après les avoir configurés correctement. Il faut ainsi configurer la valeur minimale et la valeur maximale que ce composant peut prendre.

Tâche 2

1. Depuis votre interface graphique, vous devez créer un tableau Java à fournir à `GarageManager` pour que cette classe vous génère la facture en HTML. Implémentez ce code et testez-le en l'affichant dans une nouvelle fenêtre.

Partie 4 - Impression

Vous devez dans cette partie vous débrouiller par vous-même. Nous estimons en effet que vous êtes en effet prêt-e-s à agir seul-e-s pour afin de résoudre un problème concret que nous n'avons pas vu ensemble (ce qui vous arrivera continuellement dans votre travail futur).

Tâche 1

La première tâche que vous avez est de trouver comment imprimer les factures sur une imprimante. Il doit être possible de choisir, comme dans n'importe quel programme, sur quelle imprimante on souhaite imprimer la facture etc... *Hint*: recherchez un lien entre Swing et l'impression (printing support).

Tâche 2

Vous pouvez modifier le code de la classe `GarageManager` générant les factures afin de prendre en compte d'informations comme le nom du client, générer des tableaux avec des couleurs etc… Vous pouvez également modifier l'interface graphique afin de pouvoir prendre d'autres informations que vous jugerez utiles.

Note importante :

Pour cette partie, vous êtes libres d'interpréter les consignes ci-dessus librement. Vous avez droit pour cela à utiliser toutes les ressources possibles. Vous êtes également libres de faire l'interface graphique qu'il vous plaira.

Préparation de la séance de présentation

Vous allez devoir présenter et défendre votre projet durant une séance de laboratoire un peu spéciale.

Déroulement de la présentation

Durant cette présentation, qui se déroulera devant votre ordinateur, vous allez devoir nous présenter votre code et nous vous poserons des questions sur celui-ci. Vous serez ainsi interrogés spécifiquement sur le code et sur les choix que vous avez pris dans celui-ci. Il est également possible que nous vous posions des questions générales portant sur *Java*.

Comme d'habitude pour les laboratoires, vous avez la possibilité de travailler par groupes. Dans ce cas, les questions seront posées à chacun des membres du groupe.

Notation de la présentation

Votre présentation sera intégrée au bonus comme environ 2/3 de celui-ci. La note finale du bonus vous sera communiquée ultérieurement.

Comme nous passerons du temps à regarder votre code, n'oubliez pas de le commenter de manière intelligente ! Cela aura l'avantage que vous pourrez vous retrouver plus facilement dans votre code. N'oubliez pas également qu'il est possible mettre en forme très facilement le code en pressant simultanément les touches Ctrl+Shift+f ou en allant dans le menu d'Eclipse sous Source ‣ Format.