Java et Algorithmique

Projet Jeux de plateaux :

Tic - Tac - Toe

Itération 1 – Afficher le plateau de jeu :

Pour une classe il est préférable de s'organiser ainsi :

-attributs

-constructeurs

-méthodes

-getter/setter

parseInt :

Résumé

Dans ce lab, nous avons exploré trois méthodes différentes pour valider les entrées d'entiers en Java :

1. La méthode `Integer.parseInt()`, qui est simple mais nécessite une gestion des exceptions (excepti on handling)
2. La méthode `Scanner.hasNextInt()`, qui est excellente pour la lecture des entrées utilisateur
3. La méthode `Character.isDigit()`, qui fournit une validation caractère par caractère

Chaque méthode a ses avantages et ses cas d'utilisation. La méthode `parseInt()` est bonne pour une conversion simple de chaîne de caractères en entier, `Scanner.hasNextInt()` est excellente pour la lecture des entrées utilisateur, et `Character.isDigit()` offre le plus de contrôle sur le processus de validation.

Ces techniques de validation sont essentielles pour développer des applications Java robustes qui peuvent gérer les entrées utilisateur de manière sûre et efficace.

Delegate :

En programmation Java, **l'héritage** est le processus par lequel une classe prend les propriétés d'une autre classe. Autrement dit, les nouvelles classes, appelées classes dérivées ou classes enfants, reprennent les attributs et le comportement des classes préexistantes, appelées classes de base ou classes parentes.
La délégation consiste simplement à transférer une tâche à quelqu'un ou quelque chose d'autre.

- La délégation peut être une alternative à l'héritage.
- La délégation signifie que vous utilisez un objet d'une autre classe comme variable d'instance et que vous transmettez des messages à l'instance.
- C'est mieux que l'héritage dans de nombreux cas car cela vous oblige à réfléchir à chaque message que vous transmettez, car l'instance est d'une classe connue, plutôt que d'une nouvelle classe, et parce que cela ne vous oblige pas à accepter toutes les méthodes de la super classe : vous ne pouvez fournir que les méthodes qui ont vraiment du sens.
- La délégation peut être considérée comme une relation entre des objets où un objet transmet certains appels de méthode à un autre objet, appelé son délégué.
- Le principal avantage de la délégation réside dans sa flexibilité d'exécution : le délégué peut être facilement modifié à l'exécution. Cependant, contrairement à l'héritage, la délégation n'est pas directement prise en charge par la plupart des langages orientés objet courants et ne facilite pas le polymorphisme dynamique .

Label : Un Labelobjet est un composant permettant de placer du texte dans un conteneur. Une étiquette affiche une seule ligne de texte en lecture seule. Ce texte peut être modifié par l'application, mais l'utilisateur ne peut pas le modifier directement.

raccourci pour la génération de code : alt + insert (template de code)

Mettre en place IDE Intellij pour pouvoir Run depuis n'importe qu'elle classe :

1 → Run puis Edit configurations

2 → Clique sur + puis Application

3 → Donne un nom générique comme RunCurrentFile

4 → Dans le champ Main class clique sur ...

5 → Modify Options → Add VM/Options / Program Arguments

Ctrl + P = affichage des paramètres de la fonction

.trim() → supprime tous les espaces au début et à la fin d'un String

Un **package** en Java est un dossier logique qui sert à organiser le code. Il permet de regrouper les classes par thème ou par fonctionnalité, d'éviter les conflits de noms et de garder un projet clair et structuré. On le déclare en haut du fichier avec la ligne package mon.package; et on importe les classes d'autres packages avec import mon.autrepackage.MaClasse;. En résumé, les packages sont la base d'une bonne architecture : ils permettent de ranger son code proprement et de s'y retrouver facilement dans les gros projets.

Une **classe abstraite**, quant à elle, est une classe qu'on ne peut pas instancier directement — elle sert de modèle commun pour d'autres classes. Elle peut contenir des attributs, des méthodes avec un corps, mais aussi des méthodes abstraites (sans implémentation) que les sous-classes devront obligatoirement définir. On l'utilise quand plusieurs classes partagent un comportement ou des propriétés similaires, mais doivent aussi personnaliser certaines actions. Par exemple, une classe Animal pourrait définir une méthode abstraite faireDuBruit() que chaque animal implémentera à sa façon. Cela permet de factoriser le code tout en imposant une structure claire.

Enfin, une **classe abstraite** se distingue d'une **interface** : la première fournit une base commune avec du code partagé, tandis que la seconde définit simplement un contrat de méthodes que les classes doivent implémenter. Une classe ne peut hériter que d'une seule classe abstraite, mais elle peut implémenter plusieurs interfaces.

👉 En résumé : **les packages organisent**, **les classes abstraites structurent**, et **les interfaces contractent**.

@Override sert à **indiquer que tu réécris une méthode d'une superclasse ou d'une interface**.

La méthode equalsIgnoreCase() renvoie une valeur booléenne. Si l'argument n'est pas nul et que le contenu est le même, en ignorant la casse, il renvoie *true* . Sinon *faux* .

En Java, une **exception** est un événement qui perturbe le flux normal d'exécution d'un programme. Quand une situation anormale survient (ex. : division par zéro, accès à un fichier inexistant), Java **lève (throw)** une exception pour signaler l'erreur.

Les exceptions peuvent être **prédéfinies** (ex. : NullPointerException, IOException) ou **personnalisées** (en étendant Exception ou RuntimeException).

16:51

En Java, **lever une exception** (ou *thrown exception*) signifie **signalier explicitement une erreur ou une situation anormale** dans le code à l'aide du mot-clé throw. Quand une méthode rencontre un problème qu'elle ne peut pas résoudre (par exemple, une valeur invalide, un fichier introuvable, ou une violation de règle métier), elle **crée un objet exception** (comme IllegalArgumentException, IOException, etc.) et le **lève** avec throw. Cela interrompt immédiatement l'exécution normale du bloc de code et transfère le contrôle au premier bloc catch compatible dans la pile d'appels, ou termine le programme si aucune gestion n'est prévue. L'objectif est d'**avertir** les parties du programme qui appellent cette méthode qu'un problème est survenu, afin qu'elles puissent le traiter (par exemple, afficher un message d'erreur, corriger l'entrée, ou annuler une opération). Les exceptions levées peuvent être **prédéfinies** par Java ou **personnalisées** par le développeur pour des besoins spécifiques.

throw new TypeException("Message d'erreur");

Type d’exception	Quand l’utiliser ?	Exemple
Prédéfinies	Pour des erreurs courantes déjà couvertes par Java.	IllegalArgumentException, NullPointerException
Personnalisées	Pour des erreurs spécifiques à ton application (héritent de Exception ou RuntimeException).	SoldeInsuffisantException
	Si l’appelant **doit** gérer l’erreur (ex. : E/S, base de données).	IOException, SQLException
Unchecked (Runtime)	Pour des erreurs de logique ou de programmation (gestion optionnelle).	IllegalStateException

- **Niveau d’abstraction** : Lever une exception adaptée au contexte (ex. : CompteBloqueException plutôt que Exception générique).
- **Documentation** : Utilise @throws dans la Javadoc pour indiquer les exceptions levées par une méthode.