



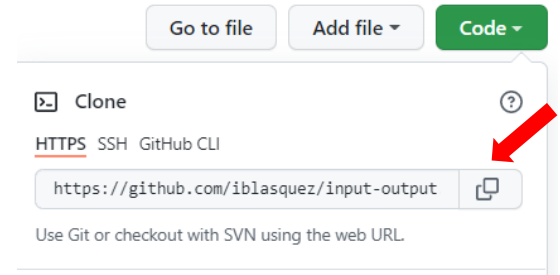
R2.03 : Des branches de chocolat exceptionnelles !



1. Récupération d'un projet sur le remote (Github)

Sur le remote :

- ☐ Rendez-vous sur github à l'adresse suivante :
<https://github.com/iblasquez/input-output>
- ☐ Cliquez sur le bouton vert **Code** et copiez le lien **https** du dépôt en cliquant sur l'icône des 2 carrés à droite de l'adresse.



En local :

- ☐ Ouvrez **Git Bash** (qui est l'invite de commande git, plus adaptée pour faire du git que l'invite de commande de Windows... attention cette invite de commande ne comprend que les commandes linux puisqu'elle est fournie avec git 😊)
- ☐ Placez-vous dans le **workspace javabut1** (les slashes des chemins sont inversés en linux et Windows)
- ☐ Clonez le dépôt **input-output** via la **ligne de commande git clone** et de l'adresse **https** copiée.

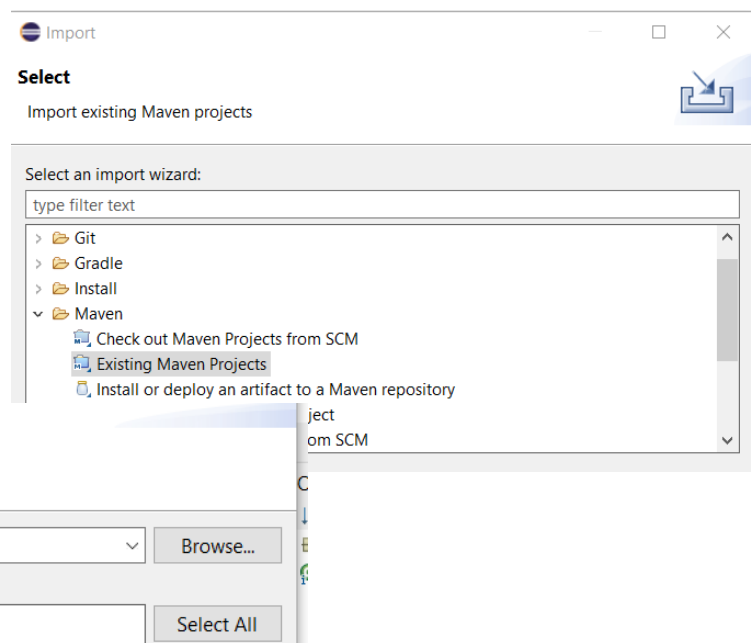
```
MINGW64:/c/Users/Isabelle/javabut1

Isabelle@XPS-15-ISA MINGW64 ~ (master)
$ cd javabut1

Isabelle@XPS-15-ISA MINGW64 ~/javabut1 (master)
$ git clone https://github.com/iblasquez/input-output.git
```

Sous Eclipse :

- ☐ Lancez Eclipse avec le workspace **javabut1**
 - ☐ Importez un **projet Maven existant** et cliquez sur **Next**.
 - ☐ Cliquez sur **Browse** et sélectionnez le projet **input-output** de manière à ce que dans le cadre **Projects** vous voyez bien un **pom.xml**
- Si tel est le cas, cliquez alors sur **Finish**



Maven Projects

Select Maven projects

Root Directory: C:\Users\Isabelle\saebut1\input-output Browse...

Projects:

☒ /pom.xml fr.unilim.iutinput-output:0.0.1-SNAPSHOT:jar Select All

2. Familiarisez-vous avec ce projet existant ...

Ce projet est le travail d'un stagiaire à qui l'encadrant a demandé d'implémenter quelques services pour pouvoir plus facilement lire et écrire des données sur la console, et notamment pouvoir faire des menus paramétrables. Ce stagiaire a appliqué les bonnes pratiques qui lui ont été enseignées :

- **Commits très fréquents** (peut-être un peu trop même) mais qui permettent de bien suivre son raisonnement et l'historique de l'ajout de nouvelles fonctionnalités à son application.
- **Travail sur la lisibilité du code** : d'une part sur le **nommage** des variables et méthodes (naming) et d'autre part sur la **décomposition du code en nombre de lignes minimales** en extrayant des méthodes pour alléger la charge mentale à la lecture du code, éviter la duplication, faciliter la réutilisation du code et la correction de potentielles erreurs
- **Répartition des responsabilités** lecture et écriture en créant une classe **ReadConsole** et **DisplayConsole** qui héritent de **Console**. **Console** permet ainsi regrouper le code commun et fréquemment utilisé par ces deux classes comme les méthodes **title** ou **message**.
- En accord avec son encadrant, il n'a pas écrit de **tests automatisés** car ce code, pour l'instant, ne s'y pas vraiment... Par contre, comme il est indispensable de vérifier et valider le « bon » comportement de l'implémentation, **des jeux d'essais** ont été écrits dans des **main**.
- **Utilisation de SonarLint** pour optimiser et améliorer la qualité son code, par exemple en transformant par exemple le **String** dans les for des méthodes **menu(...)** en **StringBuilder**

Pour vous familiarisez avec ce projet, vous allez :

❑ commencer par consulter **l'historique des versions (Team→Show In History)** qui peut vous aider à comprendre le raisonnement du développeur précédent et savoir ce qu'il vous reste à faire...

L'historique, si les commits ont bien été nommés, joue le rôle de « trace » de la phase d'implémentation.

En lisant cet historique, vous devriez comprendre :

Id	Message
17d5a5f	master (HEAD) rename TODO with git branching naming convention
d37c739	add TODO tags to improve the quality of code
57c5382	add ChocolateApplication
e4f1d7d	add answerChocolateMenuFor and emojis
b2d7c99	rename SandBoxConsole add ReadConsole with ensureIntegerBetween 2 values
8f5bfb6	add menu with hashmap as input parameter
dbc02a9	add method menu in DisplayConsole with varargs as input parameters
c0571ff	add DisplayConsole to extract showChocolateMenu
8d87737	add simple chocolate menu
40d892a	add class Console
56314cf	initial commit

- que le stagiaire « a d'abord joué » avec différentes implémentations de menu, puis a finalisé le processus autour du menu en proposant un choix entre deux valeurs et une réponse à ce choix : le tout dans la classe **SandBoxConsole**.
- Pâques approchant, un client souhaitant faire développer une application autour des chocolats s'est rapproché de l'entreprise qui avait pris en stage notre petit stagiaire 😊. Le stagiaire a donc ensuite créé la classe **ChocolateApplication** pour implémenter proprement le menu. Correspondant au menu du client.
- Malheureusement, le stage s'est terminé alors que notre stagiaire avait encore de deux ou trois bonnes idées d'améliorations et d'optimisations son programme. Heureusement avant de partir, il a noté ces idées dans son code à l'aide de **//TODO** pour que le développeur suivant puisse les implémenter...

❑ puis, **vous vous « promènerez » dans le code** pour vous l'approprier. ... Mais patience, pour l'instant vous observez et essayez juste de le comprendre, attendez les consignes avant de toucher au code 😊

❑ vous **lancerez les jeux d'essais et tests**: deux exécutables (*main*) dans le paquetage application, pas de tests automatisés : ce projet ne s'y prête pas vraiment...

❑ vous terminerez par **lancer SonarLint**: remarquez au passage que SonarLint vous indique les **//TODO** que vous pouvez également retrouver à partir du menu **Window → Show view → Tasks** (comme nous vous l'avions montré lors du premier TP)

Le stagiaire a donc laissé un programme fonctionnel (c-a-d qui marche !)

Vous ne voulez pas « casser » un programme fonctionnel, n'est-ce pas ?

Vous décidez donc d'implémenter les **//TODO** dans un (ou plusieurs branches) que vous mergerez dans *master* une fois qu'elles seront opérationnelles...

Par chance, dans son dernier commit, il vous a même donné des indices pour savoir comment nommer ces branches et à quelles tâches elles correspondaient 😊

3. Quelques mots sur les branches...

Les branches vous ont été présentées dans le cours git que vous pouvez retrouver sur <https://github.com/iblasquez/enseignement-but1-developpement>

La branche principale sur laquelle vous committez en local s'appelle **master**.

Il est possible d'ajouter à son historique d'autres branches.

Une branche permet par exemple à plusieurs développeurs différents de développer chacun des fonctionnalités (*features*) différentes et après de merger (regrouper) leur code dans *master*. Mais ce n'est pas le seul cas d'usage pour lequel vous pouvez avoir envie d'utiliser une branche (sans toucher au code qui marche).

Voici différentes « tâches » du développeur lors de la phase d'implémentation qui mènent en principe à la création d'une branche, notamment lors d'un développement collaboratif pour que chacun puisse développer dans son coin sans « casser » le code existant :

- l'ajout d'une nouvelle fonctionnalité : on parle alors de **feature**
- la correction d'un bug : on parle alors de **bugfix**
- la correction d'un bug critique: on parle alors de **hotfix**
- le nettoyage du code/amélioration de la qualité du code:
on parle alors de **refacto(ring)** ou **chore**
- l'expérimentation de fonctionnalités : on parle alors d'**experiment**
⇒ à partir de maintenant, plus de classe **SandBox** dans le code, mais une branche **experiment** que vous pourrez laisser vivre (ou pas) dans votre historique ...
- un travail en cours : on parle alors **wip** pour (**W**ork **I**n **P**rogress)

Le nom de la branche décrit succinctement le but de celle-ci. Certaines règles doivent être respectées :

- Le nom doit faire moins de 50 caractères;
- Le nom doit respecter la convention **kebab-case** (les mots doivent être en minuscule et liés par des tirets "-");
- Le nom de la branche suivant la tâche à laquelle il correspond devrait commencer par un des mots clés en gras ci-dessus

Pour en savoir plus sur la convention de nommage des branches sous git :

<https://codingsight.com/git-branching-naming-convention-best-practices/>

4. Votre première branche : **bugfix-ensure-integer-input**

Quelle est la première tâche à faire pour continuer l'implémentation de ce projet ?

Consultez la liste des **//TODO** en ouvrant **la vue Tasks dans votre IDE** :

Tasks ×					
3 items					
!	Description	Resource	Path	Location	Type
	TODO bugfix: only an integer catch the exception	ReadConsole.java	/input-output/src/main/java/io/gui/console	line 6	Java Task
	TODO feature: display the content of the file easter.txt in s...	ChocolateApplication.java	/input-output/src/main/java/io/application	line 59	Java Task
	TODO refactor: use an HashMap or something else ?	ChocolateApplication.java	/input-output/src/main/java/io/application	line 19	Java Task

Trois tâches différentes se trouvent dans les **//TODO** :

- une correction d'un bug (**bugfix**)
- un ajout de fonctionnalité (**feature**)
- un nettoyage du code pour améliorer de la qualité de code (**refacto**)

Dans quel ordre traiter ces tâches ?

Mon conseil :

Il faut tout d'abord commencer par traiter les **bugs** éventuels pour que le programme ne plante « jamais ».

Une fois, les bugs détectés corrigés, il est possible d'envisager l'ajout d'une nouvelle **fonctionnalité** c-a-d apporter un petit peu plus de valeur métier à l'application.

Dans un développement, il faut également se laisser du temps de temps en temps pour **nettoyer le code et améliorer sa qualité**.

Nous traiterons donc les tâches dans l'ordre dans lequel elles apparaissent ici dans la vue Tasks c-a-d **bugfix**, puis **feature**, puis **refacto** 😊

Pour implémenter le **bugfix**, nous choisissons donc de créer une branche, que nous appellerons (en adéquation à la convention de nommage des branches git)

bugfix-ensure-integer-input

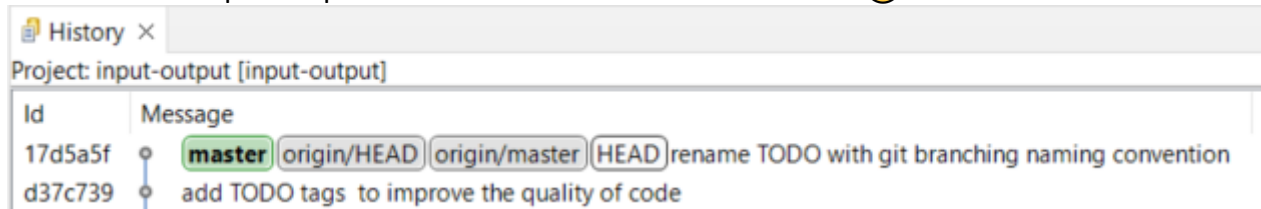
❑ a. Création de la branche bugfix-ensure-integer-input

Si vous ne voyez pas l'historique git dans votre IDE, ouvrez-le : **Team→Show In History**

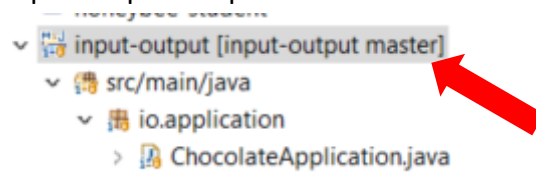
Vérifiez que le code que vous voyez actuellement est bien celui du dernier commit. C'est bien le cas si :

→ **dans la vue History** : les tags **master** et **HEAD** se trouvent sur le même commit (en l'occurrence le dernier commit de master).

Si vous êtes connecté à un remote les tags origin/HEAD et origin/master apparaîtront également (origin qui par convention est le remote origin : voir cours !). Vous êtes actuellement connecté à mon remote input-output de Github car vous avez fait un clone 😊



→ **dans la vue Package Explorer** : les crochets après le nom du projet indiquent bien master et non un hash de commit qui indiquerait que le commit de l'historique en train d'être visualisé

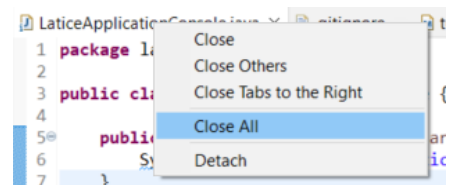


Si vous êtes sur le dernier commit, vous pouvez continuer.

Sinon positionnez-vous sur le dernier commit grâce à **Team→ Switch To →master**

Commencez par fermer tous les onglets dans l'éditeur de code.

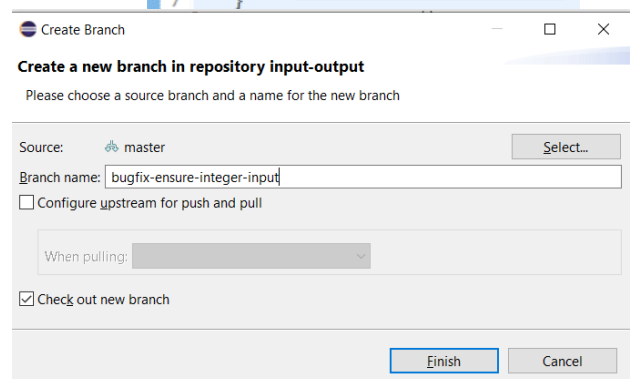
Placez-vous sur l'onglet d'un fichier de code et faites **Close All**.



Depuis le projet **input-output**
dans la vue **Package Explorer**, choisissez maintenant :
Team→ Switch To →new Branch

⇒ Vérifiez bien que la **Source** est **master** car nous voulons que la branche parte du dernier commit de master.

⇒ Entrez dans **Branch Name** le nom de la branche **bugfix-ensure-integer-input** (respect des conventions de nommage)



⇒ Vérifiez bien que **Check out new branch** est bien **coché** (voir cours : création de la branche et positionnement sur la branche 😊)

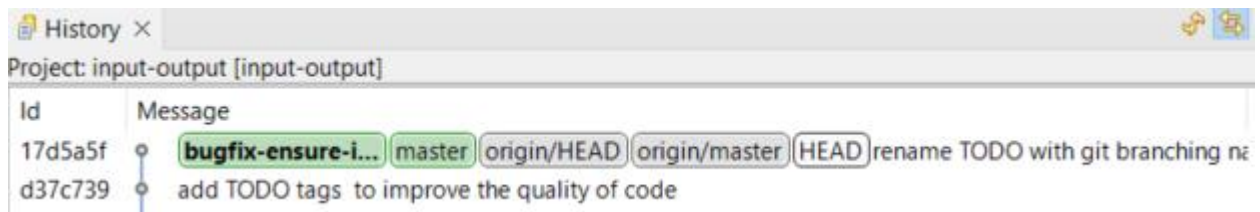
⇒ Il ne reste plus qu'à cliquer sur **Finish** !

Si tout c'est bien passé :

- Dans la **vue Package Explorer** le nom de la branche apparaît à côté du nom du projet vous indiquant que vous êtes maintenant situé sur cette branche

```
✓ > input-output [input-output bugfix-ensure-integer-input]
> src/main/java
```

- Dans la **vue History** un tag contenant le nom de la branche **bugfix-ensure-integer-input** apparaît maintenant **en vert** vous indiquant également que vous êtes sur cette branche.



Faites en sorte de bien être positionné sur la nouvelle branche créée **bugfix-ensure-integer-input** pour pouvoir continuer !!!

Remarque : Nous avons créé cette branche directement dans l'IDE, mais comme pour toute commande git vous pouvez également effectuer cette action en ligne de commande ou avec un outil visuel tel que GitKraken, Git Extension 😊

❑ b. Correction du(des) bug(s) identifié(s)

⇒ Revenir sur la **vue Tasks** et double-cliquez sur le **TODO bugfix** pour vous retrouver dans le bout de code qui doit être **fixé** c-a-d là où le stagiaire a mis son marqueur TODO indiquant qu'il vous faut intervenir ici pour corriger le bug qu'il a détecté, mais qu'il n'a pas eu le temps de corriger 😊. Vous devriez vous retrouver dans la méthode `ensureInteger` de la classe `ReadConsole`

⇒ Pour bien comprendre ce bug, vous allez lancer les jeux d'essais relatifs à l'application sur le Chocolat c-a-d exécuter le main de **ChocolateApplication**. Vérifiez tout d'abord que les cas « normaux » fonctionnent correctement, en relançant plusieurs fois ce main et en jouant les jeux d'essais suivants (complétez le tableau)

Your choice	Affichage obtenu sur la console
1	
2	
3	
0	
a	

Remarque : Peut-être pouvez-vous comprendre ici **l'intérêt de l'automatisation des tests en terme de gain de temps** pour vérifier un comportement plutôt que de devoir lancer les jeux d'essais à la main avec des exemples/valeurs différent(e)s à chaque fois 😊

⇒ on aurait vérifié plus vite avec des tests automatisés et JUnit 😊

⇒ **Corrigez le bug** en vous aidant des tests que vous venez d'effectuer manuellement (sous forme de jeux d'essais) et des résultats que vous avez noté dans le tableau.

- Quelle est ce bug et comment se matérialise-t-il ?
- **Faites-en sorte d'attraper dans la méthode ensureInteger la « bonne » exception qui évitera à votre application de planter et de traiter ce bug dans cette méthode, c-a-d ...**
- **... faites également en sorte de ne pas sortir de la méthode ensureInteger tant qu'un entier n'aura pas été saisi et effectuer uniquement cette vérification.** Un message d'erreur du type : **It must be an integer** pourra être affiché sur la console pour indiquer clairement à l'utilisateur quelle a été son erreur.

Rappel : la validité de l'entier dans l'intervalle des valeurs se traitent dans la méthode qui appelle ensureIntegerBetween, ici on a bien séparer les responsabilités : chaque méthode a la sienne 😊

⇒ **Vérifiez la non-régression de votre application** c-a-d que le code que vous venez d'ajouter pour corriger le bug n'a pas modifié le comportement initial du programme.

Pour cela, vous devez refaire les tests précédents, en vous assurant que les comportements sont identiques et que le nouveau comportement est également bien présent...

Your choice	Affichage obtenu sur la console
1	
2	
3	
0	
a	

Remarque : Peut-être comprenez-vous ici l'intérêt de l'automatisation des tests en terme de **non-régression** c-a-d qu'il faut toujours vérifier que ce que nous venons d'ajouter, en plus d'avoir le bon comportement, n'a eu aucun impact sur le comportement initial (c-a-d ce qui était déjà implémenté avant notre intervention et fonctionnait correctement doit continuer à fonctionner de la même manière)

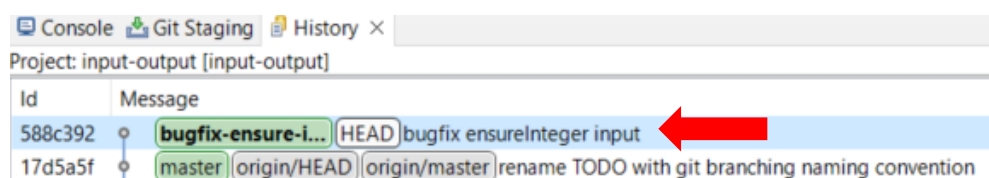
⇒ **une fois écrit, les tests automatisés garantissent la non-régression de l'application c-a-d le même comportement à chaque exécution** et la vérification se fait plus vite avec des tests automatisés et Infinitest qu'en lançant des **main** 😊

⇒ **N'oubliez pas de commiter** tout au long de la correction de votre bug.

Notre bug est tout petit ici, il ne nécessite qu'un seul commit par exemple avec le message :

bugfix ensureInteger input

Dans la **vue History**, vous devez voir votre dernier commit (HEAD) qui vient d'être effectuée sur la branche **bugfix-ensure-integer-input** : c'est pour cela que les marqueurs **HEAD** et **bugfix-ensure-integer-input** (en vert) se trouvent désormais sur la même ligne et que si nous faisons un nouveau commit il se fera à nouveau sur la branche car HEAD est désormais bien associé à la branche....

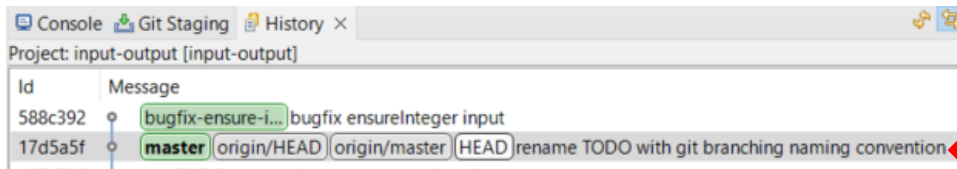


❑ c. Merger la branche dans master

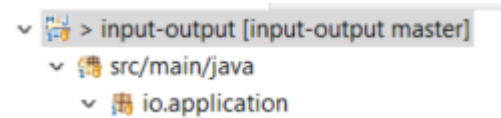
Lorsque vous avez corrigé le bug, vérifié et validé le nouveau comportement, vérifié la non-régression du code existant et effectué tous le(s) commit(s) nécessaires pour prendre en compte ces modifications, vous êtes prêts à **merger** votre branche (travail) dans master.

Pour merger la branche **bugfix-...** dans la branche **master**, procédez comme précédemment :

⇒ placez-vous sur **la branche qui doit recevoir la fusion** c-a-d qu'il faut revenir sur **master** via un **Team** → **Switch to** → **master** et vérifier que le tag HEAD se trouve désormais sur le même commit que le tag **master** (en vert)



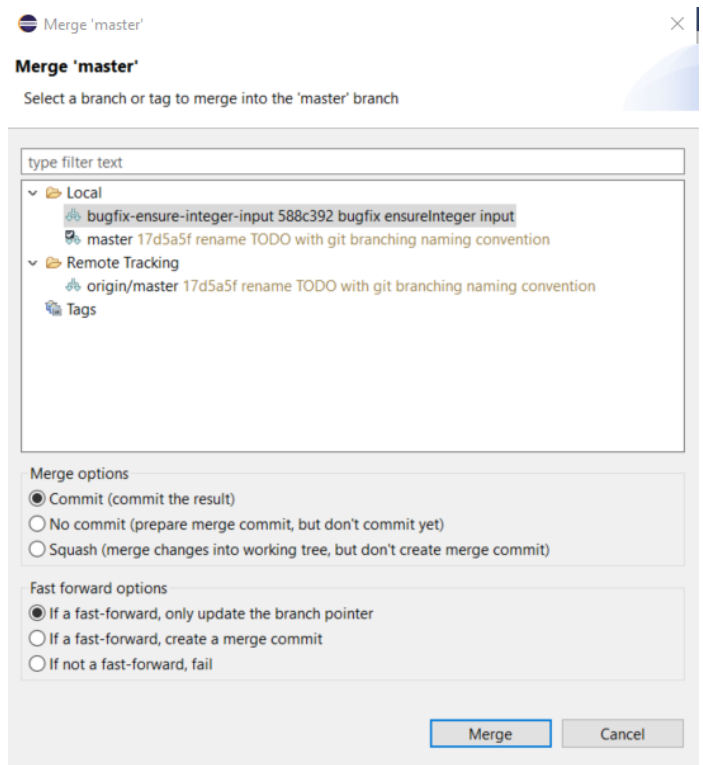
et/ou que la vue package explorer indique également **master** après le nom du projet :



⇒ depuis la vue Package Explorer, sélectionnez l'option de **fusion** via un clic droit et **Team** -> **merge**

⇒ sélectionnez la branche à fusionner dans master : **bugfix-...** (dans **Local** bien sûr)

⇒ conservez les options par défaut (**commit** et **fast-forward**) et cliquez sur **Merge** (ou simplement double-cliquez sur la branche à merger 😊)

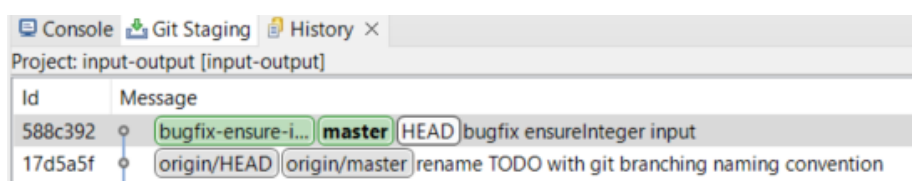


Une fenêtre **Merge Result** s'ouvre indiquant que la fusion s'est bien passée en indiquant que le **résultat** est bien **Fast-forward**.

Il ne reste plus qu'à cliquer sur **OK**...



... et consulter l'historique : les tags **master** et **bugfix-...** sont revenus sur le même commit, donc la fusion s'est bien passée !



Remarque : Il s'agit ici d'une **fusion fast-forward** car l'historique n'avait pas évolué depuis la création de la branche, mais bien souvent dans un développement collaboratif master évolue pendant que vous travaillez sur une branche et lors de la fusion un **commit de fusion** est créé et un **conflit** peut également apparaître (à résoudre pour finaliser la fusion) comme cela vous a été présenté dans le cours...et vous en ferez sûrement l'expérience lors de votre prochain développement en mode collaboratif 😊

Est-il possible de se déconnecter d'un remote pour continuer à développer seulement en local ?

Oui, cela est tout à fait possible et dans cet exercice, il n'y a aucun intérêt à ce que vous gardiez le remote vers mon github car vous n'allez pas tous pousser vers mon github 😊
Nous allons donc supprimer la connexion vers le remote en **ligne de commande**.

Ouvrir une console **git bash** et placez-vous dans le **projet input-output** du workspace javabut1.
Tapez la commande : **git remote -v**

```
$ git remote -v
origin https://github.com/iblasquez/input-output.git (fetch)
origin https://github.com/iblasquez/input-output.git (push)
```

Cette commande permet de lister les connexions au(x) remote(s) (et oui, nous pourrions nous connecter à plusieurs, mais vous ne le ferez pas pour le moment).

Actuellement ce dépôt est lié à un remote nommé **origin** dont l'adresse utilisée :

- **pour tirer le code (pull = fetch + merge)** est : <https://github.com/iblasquez/input-output.git>
- **pour pousser le code (push)** est : <https://github.com/iblasquez/input-output.git>

Pour déconnecter votre dépôt local d'un remote, vous devez utiliser la commande :

git remote rm <nom du remote>

Ici ce sera donc :

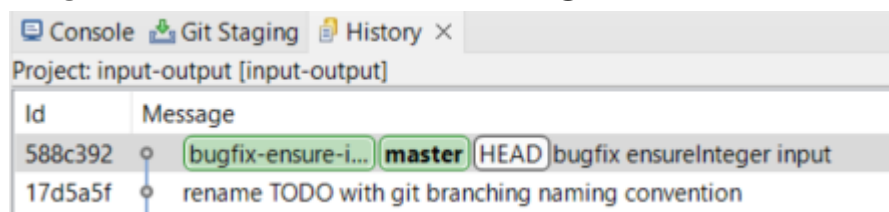
```
$ git remote rm origin
```

... puis vous redemanderez un

```
$ git remote -v
```

 pour constater qu'origin a disparu...

Rendez-vous maintenant dans la vue History d'Eclipse pour constater également que les tags gris **origin/HEAD** et **origin/master** ont également disparu 😊



Si vous en avez envie, vous pouvez, en fin de TP, connecter votre dépôt local input-output à un de vos remote (github, gitlab,...) et le pousser.

5. Ajout d'une nouvelle fonctionnalité : afficher le contenu du fichier `easter.txt`

Nous allons maintenant nous occuper du deuxième **TODO** à traiter.

Ouvrez la **vue Tasks** pour vous rendre plus facilement à l'endroit dans le code où se **TODO** a été spécifié.

Le `//TODO` explicite bien le travail à effectuer.

Vérifiez qu'un fichier **easter.txt** se trouve bien dans le répertoire `src/main/resources`

❑ a. Création de la branche `feature-show-file-easter-txt`

A partir du dernier commit de master, créer une branche : **feature-show-file-easter-txt** et placez-vous sur cette branche pour commencer à implémenter cette nouvelle fonctionnalité.

❑ b. Implémentation dans la branche `feature-show-file-easter-txt`

⇒ **Quoi implémenter et où ?**

Montrer le contenu du fichier **easter.txt** revient à afficher le contenu de ce fichier sur la console.

Il semblerait donc normal que la nouvelle fonctionnalité soit implémentée dans la classe

DisplayConsole :

- Ajoutez donc à la classe **DisplayConsole** une méthode du type :

```
public String showContentFile(String nameFile) {
    StringBuilder content = new StringBuilder();
    try (FileReader reader = new FileReader(nameFile)) {
        int character;
        while ((character = reader.read()) != -1) {
            content.append((char) character);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return content.toString();
}
```

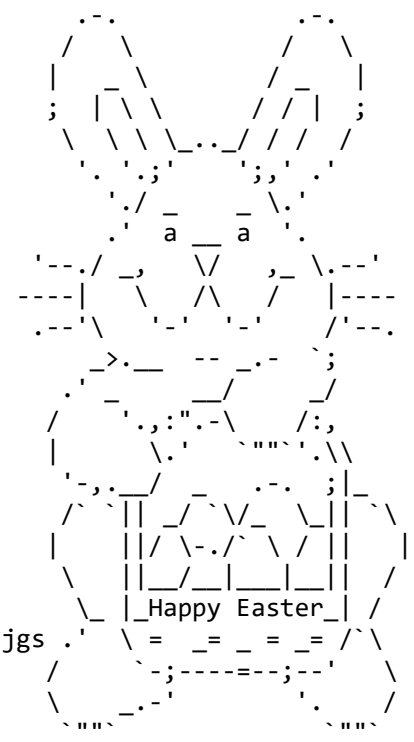
Si vous avez bien visionné les vidéos, vous reconnaîtrez aisément ce code qui permet de lire caractère par caractère le contenu du fichier dont le nom est `nameFile` 😊

- ... Et appelez cette méthode à l'emplacement du `//TODO`.
En voyant votre code, votre encadrant vous indique que le stagiaire s'est trompé et que c'est **pour le chocolat au lait que le client souhaite afficher le fichier** 😊
Votre méthode `answerChocolateMenuFor` doit donc être modifiée de la manière suivante :

```
private void answerChocolateMenuFor(int option) {
    switch (option) {
        case 1:
            // code existant ...
        case 2:
            console.message(display.showContentFile("src/main/resources/easter.txt"));
            // code existant ...
        case 3:
            // code existant ...
        default:
            // code existant ...
    }
}
```

Implémentez la méthode `showContentFile` de la classe **DisplayConsole** de manière à ce que vos jeux d'essais aient les comportements du tableau suivant :

Tableau 1 :
Tableau d'exemples illustrant le comportement de l'application Chocolat :

Your choice	Comportement attendu : affichage obtenu sur la console
1	Healthy chocolate !🍫👍
2	Art by Joan Stark (https://www.asciart.eu/holiday-and-events/easter)  The best chocolate for easter !🐰
3	Sugar and fat : unhealthy !🚫🍫🤢
0	🙄 Maybe you dislike chocolate...
a	It must be an integer

⇒ Après avoir vérifié le comportement de votre application et passer votre code sous SonarLint, effectuez un commit pour historiser la mise en place de l’affichage du contenu du fichier `easter.txt` avec un message du type : **add showContentFile in DisplayConsole**

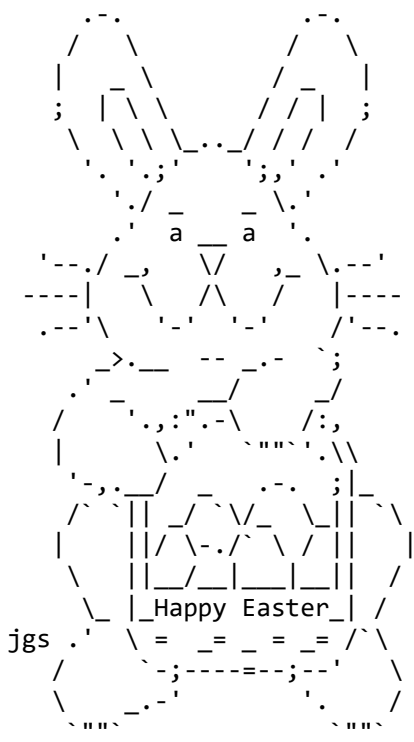
⇒ Amélioration du traitement de l'exception

... Dans le code de la méthode `showContentFile` précédente, l’exception était attrapée directement dans **catch** et la trace des erreurs était directement affichée avec l’instruction `e.printStackTrace()` ; Dans notre application, nous souhaitons que cette exception soit **transformée en exception métier** que vous appellerez **EasterEggFileException** (que vous implémenterez dans un package `io.util` ou `io.exception`) et qui sera propagée pour être attrapée au moment de l’appel de **showContentFile** (c-a-d dans le **case 3** du **switch (option)**)
 Si cette exception se propage jusque-là, à la place du fichier, le client souhaite qu’un cadeau sous forme d’emoji soit affiché (tant pis l’utilisateur ne saura pas qu’il y avait easter egg en ascii-art 🍪, mais nous saurons qu’il y a une erreur avec le cadeau)

Remarque : cherchez dans le code qui vous a été donné comment rechercher le code d’un emoji. Les stagiaire précédent vous a laissé ces informations dans son code 🍪

Tableau 2 :

Tableau illustrant des deux comportements de l'application Chocolat dans le cas d'un chocolat au lait :

Your choice	Comportement attendu : affichage obtenu sur la console
2 – fichier accessible	<p>Art by Joan Stark (https://www.asciart.eu/holiday-and-events/easter)</p>  <p>jgs . ' \ = _ = _ = _ = \ / \</p> <p>🍫 The best chocolate for easter !@</p>
2 – pb d'accès fichier au fichier <i>Changer par exemple le nom du fichier en east.txt en ouvrant un explorateur de fichier</i>	<p>📁</p> <p>🍫 The best chocolate for easter !@</p>

Avant de continuer, redonnez dans l'explorateur de fichier le bon nom au fichier `easter.txt` et refaites un jeu d'essai avec l'option 2 pour être sûr que le programme fonctionne normalement et que le lapin s'affiche bien sur la console 😊

Encore une fois, on comprend l'utilité des tests automatiques : ce n'est pas pratique de changer à la main le nom du fichier 😊 ...

Après avoir vérifié et validé l'implémentation du nouveau comportement (affichage de l'emoji cadeau lié à la capture de l'exception) il ne vous reste plus qu'à effectuer un nouveau commit avec un message du genre : **add EasterEggException and show wrapped present emoji**

❑ c. Merger la branche `feature-show-file-easter-txt`

Il ne reste plus qu'à merger ce beau travail dans la branche **master**.

A vous de jouer en vous inspirant de la fusion de la branche précédente 😊

Encore une fois, le résultat sera **fast-forward** car la branche master n'a pas été modifiée depuis la création de cette branche....

6. Restructuration du code existant (amélioration du design) et nettoyage du code

Nous allons maintenant nous occuper du dernier **TODO** à traiter.

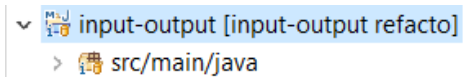
Ouvrez la **vue Tasks** pour vous rendre plus facilement à l'endroit dans le code où se **TODO** a été spécifié.

Le `//TODO` suggère un remaniement du code, nous allons donc appeler cette branche **refacto**

Un refactoring consiste à changer la structure du code sans en changer son comportement (c-a-d qu'à la fin de l'implémentation, il faudra refaire tous les tests/jeux d'essais (tableau 1 et tableau 2) pour s'assurer que le comportement de l'application n'a pas changé...C'est pour cela qu'habituellement, on couvre le code avec des tests avant un refactoring pour s'assurer que le comportement ne change pas c-a-d assurer la non-régression de l'application.

☐ a. Création de la branche refacto

A partir du dernier commit de master, créer une branche : **refacto** et placez-vous sur cette branche pour commencer le refactoring



☐ b. Restructuration du code existant (amélioration de la conception)

Le stagiaire a laissé un `//TODO` au niveau de la déclaration des codes des Emojis.

Lorsque vous avez voulu ajouté un nouvel emoji, vous avez normalement ajouté une constante.

Le stagiaire a bien senti **une mauvaise odeur (code smell) à cet endroit du code...**

Comme il venait d'utiliser un HashMap pour le menu, il s'est dit pourquoi pas faire pareil, au moins on centralisera tous les emojis dans une même méthode.

Pensez-vous que ce soit la meilleure solution et pourquoi ?

.....

.....

.....

.....

.....

Avec l'expérience et les exemples que vous avez pu implémenter au cours de ces dernières semaine, quelle autre solution pourriez-vous proposer ?

.....

.....

.....

.....

.....

Lorsque vous vous lancez dans un refactoring, il faut prendre le temps d'envisager plusieurs solutions pour la restructuration du code et de choisir celle qui sera la plus simple à implémenter, maintenir et étendre dans le contexte de notre application :

On fit qu'il faut rester **KISS** (Keep it simple, stupid)

.. et surtout poser-vous toujours **la question de la Responsabilité** :

Est-ce que c'est de la responsabilité de la classe `ChocolateApplication` de gérer les emojis ?

☐ OUI

☐ NON

Nous pensons que ce n'est pas de la responsabilité de la classe `ChocolateApplication` de gérer les codes des émojis ⇒ il faut donc déplacer cette responsabilité dans une nouvelle classe.

Lors de l'implémentation de la carte à jouer, la **Suit** gérait des émojis.

Retrouvez ce code dans vos implémentations passées.

Suit est en réalité une

En vous inspirant de l'implémentation de **Suit**, implémentez dans le package

application.gui.console une énumération **Emoji**.

Faites en sorte **ChocolateApplication** compile avec cette nouvelle conception et que le code correspondant à la réponse de l'option du chocolat noir ressemble désormais à :

```
console.message(Emoji.CHOCOLATE_BAR+" Healthy chocolate !" + Emoji.OK_HAND + Emoji.THUMBS_UP);
```

Une fois cette nouvelle implémentation implémentée, n'oubliez pas de supprimer toutes les constantes en haut de la classe `ChocolateApplication` 😊

Après ce refactoring, il ne vous reste plus qu'à vérifier et valider que vous n'avez « rien cassé » et que le comportement de l'application est toujours le même que précédemment : pour cela, vous devez donc rejouer les exemples des tableaux 1 et 2.

Une fois le comportement vérifié et validé, il ne vous reste plus qu'à effectuer un nouveau commit avec par exemple le message : **add enumeration Emoji**

Encore un peu de refactoring : Nettoyage du code ?

Tant que vous êtes en phase de refactoring, vous pouvez vous demander s'il existe du *code mort* dans votre application ?

Rendez-vous par exemple dans la classe **DisplayConsole**

Si vous voulez réutiliser cette classe dans un autre projet (la SAE par exemple), cette classe doit rester générique : pas de chocolat dans la SAE...

La méthode `showChocolateMenu` est spécifique à l'application que nous sommes en train d'écrire et en plus elle n'est pas utilisée dans cette application, puisque l'implémentation d'un menu paramétré a été choisi. Vous pouvez donc normalement supprimer sans problème la méthode `showChocolateMenu` qui n'est absolument plus utilisé dans votre application et est devenu du *code mort*...

Procédez à la suppression de cette méthode de la classe **DisplayConsole**.

Sauvez et Compilez. Que se pass-t-il ?

La classe **SandboxConsole** est juste un *bac à sable*, qui sert à tester des choses, à faire des expérimentations, mais qui ne devra pas être livrée avec l'application et ne doit avoir aucun impact sur l'application.

A vous de choisir comment vous voulez corriger cette erreur de compilation :

- Soit en faisant en sorte que l'implémentation de la méthode `showChocolateMenu` se retrouve maintenant comme méthode privé dans la classe `SandboxConsole` et que l'appel à cette méthode soit modifié de la sorte :

```
console.title("Menu with directly text in method ");
console.message(showChocolateMenu());
```
- Soit en supprimant les deux lignes précédentes puisque vous jugez que l'objectif de la classe `SandboxConsole` était juste à la base de faire une expérimentation sur les menus paramétrables.

A vous de voir ce que vous préférez, quel que soit votre choix, rendez la classe `Sandbox` compilable car on ne commite pas avec des erreurs de compilation !!!!

N'oubliez pas de vérifier que la suppression du code de la méthode **showChocolateMenu** dans votre application n'a eu aucun impact. Rejouez encore une fois les jeux d'essais du tableau 1 et 2 pour vérifier la non-régression 😊

Si tel est le cas, vous pouvez procéder à un nouveau commit :

```
delete showChocolateMenu from DisplayConsole
```

❑ c. Merger la branche refactor

Il ne reste plus qu'à merger ce beau travail dans la branche **master**.

A vous de jouer en vous inspirant des fusions précédentes.

Encore une fois, le résultat sera **fast-forward**.

7. Les bacs à sable sont des expérimentations 😊

Nous venons de voir que la classe **SandboxConsole** dans la branche **master** a généré un problème (erreur de compilation quand nous avons voulu supprimer du code mort de notre application).

Du moment que votre projet est versionné, les expérimentations ne doivent pas se faire dans la branche principale **master**, mais plutôt dans une branche **experiment**.

Nous allons donc repérer cette erreur et faire maintenant en sorte que la classe **SandboxConsole** se trouve dans une branche **experiment-console** (où nous pourrions toujours aller consulter le code et continuer l'expérimentation avec d'autres commits sur cette branche si le cœur nous en dit) puis nous supprimerons la classe **SandboxConsole** de la branche principale car elle n'a rien à faire dans l'application de notre client 😊

❑ Création de la branche **experiment-console**

A partir du dernier commit de **master**, créer une branche : **experiment-console**

Cette fois-ci vous n'avez pas besoin de vous placer sur cette branche car pour l'instant, on ne souhaite pas poursuivre cette implémentation. On va donc laisser la branche vivre, c-a-d que **nous ne la mergerons pas dans master !!!!**

❑ Suppression de la classe **SandboxConsole** de la branche **master**

Maintenant que nous savons que la classe **SandboxConsole** peut être retrouvée à tout moment en se plaçant sur le commit de la branche **master**, nous pouvons supprimer cette classe de la branche **master** car comme nous l'avons dit, elle n'a plus rien à faire dans notre application :

⇒ Placez-vous sur le dernier commit de **master** :

```
▼ input-output [input-output master]
```

⇒ Supprimez (delete) la classe **SandboxConsole** du package **io.application**

⇒ Vérifiez que le comportement de votre application n'a pas été impacté par cette suppression (qui n'est autre qu'un refactoring qui consiste à effacer du code)

⇒ Effectuez un nouveau commit avec le message : **delete SandboxConsole**

(bien que vous l'ayez supprimé du projet, pour que cette suppression soit bien prise en compte dans le commit, n'oubliez pas de passer la classe **SandboxConsole** dans Staged Changes avant de commiter 😊)

Bonne pratique à retenir : Pas d'expérimentation dans master avec le gestionnaire de version on passe par une branche experiment qu'on laisse vivre ou pas

Dorénavant, vous ne mettrez donc plus de classe SandBox dans master, mais vous créerez une branche dès lors que vous souhaitez faire une expérimentation !!!

Pour terminer, visualisez la vue History, vous retrouvez master et les différentes branches (puisque pour le moment vous avez choisi de ne supprimer aucune branche)

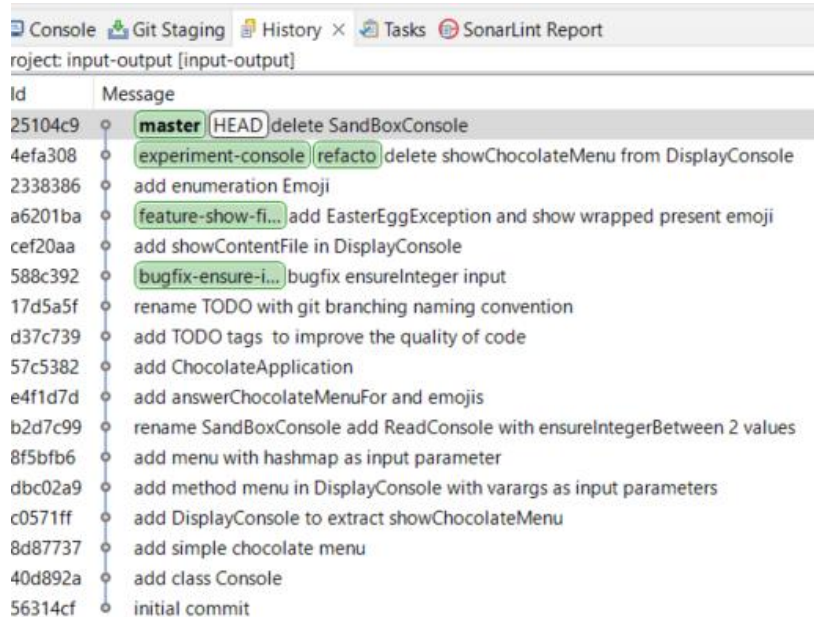
8. Pousser votre code sur un remote

Tout au long de cet exercice, vous avez uniquement effectué des commits en local sur votre machine.

Maintenant que l'exercice/la séance est terminée, il est temps de pousser ce code sur un de vos remotes pour que vous y ayez accès à tout moment de n'importe où.

Rappelez-vous, ce dépôt n'est actuellement lié à aucun remote (puisque nous l'avons déconnecté de son dépôt initial au cours du TP).

Poussez ce code sur un de vos remotes (Github, GitUnilim, Gitlab,...) et appelez votre enseignant de TP une fois cette action effectuée pour qu'il puisse jeter un petit coup d'œil sur votre code 😊



Id	Message
25104c9	delete SandBoxConsole
4efa308	delete showChocolateMenu from DisplayConsole
2338386	add enumeration Emoji
a6201ba	add EasterEggException and show wrapped present emoji
cef20aa	add showContentFile in DisplayConsole
588c392	bugfix ensureInteger input
17d5a5f	rename TODO with git branching naming convention
d37c739	add TODO tags to improve the quality of code
57c5382	add ChocolateApplication
e4f1d7d	add answerChocolateMenuFor and emojis
b2d7c99	rename SandBoxConsole add ReadConsole with ensureIntegerBetween 2 values
8f5bfb6	add menu with hashmap as input parameter
dbc02a9	add method menu in DisplayConsole with varargs as input parameters
c0571ff	add DisplayConsole to extract showChocolateMenu
8d87737	add simple chocolate menu
40d892a	add class Console
56314cf	initial commit

9. Lien possible avec la Sae :

Dans votre SAE, si vous décidez de passer par un mode console, vous pouvez ré-utiliser les fichiers **Console**, **ReadConsole**, **DisplayConsole** **Emoji** du paquetage **io.gui.console**. 😊

10. En savoir plus ...

❑ sur les fichiers en Java (Java I/O, NIO et NIO 2)

→ Les vidéos (en français) sur l'accès aux fichiers Java sur la chaine youtube [Cours-en-ligne](https://www.youtube.com/c/coursenlignejava/playlists?view=50&shelf_id=5) de [José Paumard](#) (en français) : https://www.youtube.com/c/coursenlignejava/playlists?view=50&shelf_id=5

→ Les tutoriels sur le site Baeldung (en anglais) : <https://www.baeldung.com/java-io>

Si le cœur vous en dit, vous pouvez créer un branche **experiment-file** pour essayer d'écrire un code plus concis que celui actuellement présent dans la méthode `showContentFile` 😊

❑ sur git (en lien avec ce TP) ..

<https://www.delftstack.com/fr/howto/git/how-to-remove-a-git-remote-url/>

<https://codingsight.com/git-branching-naming-convention-best-practices/>

<https://www.codeheroes.fr/2020/06/29/git-comment-nommer-ses-branches-et-ses-commits/>

<https://enlear.academy/26-git-command-i-use-all-the-time-cheatsheet-6c5682ded2af>