Aix*Marseille université
Socialement engagée

Année 2021-22

Domaine Sciences et Technologies
Portail René Descartes

Programmation 1 : TP 1
Code UE : SPO2U07L

TP 1 : Prise en main de Java et Repl.it

L'objectif de ce TP est de découvrir Java au travers de l'interface proposée par le site `repl.it`, que nous utiliserons pour tous les TP. Nous verrons comment les classes et les objets nous permettent d'organiser un programme. Java est un langage de programmation, dit orienté objet car il est basé sur la notion d'objet. `repl.it` est une interface qui permet d'écrire des programmes, de les exécuter, et d'en voir les éventuels résultats.

Il vous est aussi possible d'utiliser un environnement de développement (IDE), c'est à dire un logiciel permettant d'éditer le programme en apportant de nombreuses fonctionnalités pratiques. Ce choix doit cependant se faire à la condition que vous soyez totalement autonome avec ce mode de fonctionnement ; les consignes que nous vous fournirons concerneront uniquement `repl.it`. Parmi les IDE possibles, JetBrains, l'éditeur de l'environnement de développement pour Java IntelliJ, propose des licences gratuites pour les étudiants, il suffit de vous créer un compte étudiant sur leur site, à l'adresse `https://www.jetbrains.com/student/`.

1 Premier programme Java

1. Aller à l'adresse `https://repl.it/`.

Si vous n'en avez pas déjà un, créez vous un compte en cliquant en haut à droite sur **Sign up** et en suivant les instructions.

2. Une fois votre compte créé, créer un nouveau projet :

— Cliquer sur +**New repl** en haut à gauche,
— Pour "Language", choisir "Java"
— Pour le nom du projet, donner "TP1"

Votre fenêtre de navigation s'organise alors en trois parties˜ :
— À gauche on trouve une colonne "Files" qui comprend pour le moment un unique fichier `Main.java`.
— Au milieu on voit le contenu de ce fichier qui correspond à la classe `Main.java`.
— À droite s'affiche la console où s'afficheront les résultats, ainsi qu'un "Shell" qui permet de se retrouver dans l'arborescence des fichiers. Dans le "Shell" vous devez normalement voir s'afficher >. Si vous tapez la `ls` qui liste le contenu du répertoire et vous devriez avoir pour seul résultat `Main.java`.

3. Un programme Java est organisé en classes, chaque classe étant définie dans un fichier. Pour l'instant, vous disposez d'une unique classe `Main` qui est composée :

— D'une ligne de déclaration `class Main`.
— D'un bloc délimité par des accolades. Tout ce qui sera ajouté à la classe devra être mis entre ces deux accolades.

Pour avoir un programme exécutable, le projet doit avoir un *point d'entrée*, qui contient les instructions qui seront évaluées lors de l'exécution. En Java, les points d'entrée sont toujours déclarés de la manière suivante :

```
public static void main(String[] args) {
  // nous mettrons les instructions du programme dans ce bloc
}
```

Nous étudierons plus tard ce que veux dire cette formulation, en résumé elle signifie que nous déclarons une fonction (`static` en Java, on emploie le terme *méthode statique*), sans retour (`void`), sans restriction (`public`), et qui s'appelle `main`. La partie entre parenthèse définit les paramètres de la fonction (ici un tableau

de textes nommés `args` : `String[] args` que nous n'utiliserons pas). Ensuite, une paire d'accolades délimite les instructions de la méthode statique `main`.

Pour le moment, votre fonction ne contient qu'une ligne d'instruction

```
System.out.println("Hello world!");
```

qui doit faire afficher à la console la chaîne de caractère "Hello world !".

> Lancer votre programme en cliquant sur **Run** en haut au milieu.

4. Si tout s'est bien passé, vous devez voir s'afficher deux lignes dans la console :

```
> javac -classpath .:/run_dir/junit-4.12.jar:target/dependency/* -d . Main.java
> java -classpath .:/run_dir/junit-4.12.jar:target/dependency/* Main
```

suivies d'une ligne `Hello world !`.

La première ligne correspond à la commande ayant permis la *compilation* du programme et la deuxième à la commande ayant permis son *exécution*. Le résultat étant produit sur la dernière ligne.

Compiler le programme consiste à transformer votre programme tel que vous l'avez écrit, c'est-à-dire compréhensible par un être humain, vers un langage propre à l'ordinateur, en l'occurrence du *bytecode java*. Cette transformation est effectuée par la commande *javac*, le *compilateur*. Ainsi vous pourriez aussi compiler votre programme sur votre ordinateur, sans utiliser `repl.it`, du moment que vous avez installé un compilateur Java.

Exécuter le programme consiste à évaluer les instructions prévues par le programmeur, de façon à produire l'effet désiré (calculer les décimales de π, envoyer un courriel, lancer une partie de jeu vidéo, ...). Ce n'est pas votre programme qui est directement exécuté, mais la version compilée obtenue avec `javac`. L'exécution des instructions en bytecode java est supervisée par le programme `java`. De même vous pourriez aussi exécuter le programme sur votre ordinateur personnel, sans `repl.it`.

5. Afin de bien comprendre le fonctionnement de la compilation, on va utiliser le terminal (shell) de `repl.it`, présent comme onglet dans la partie droite de la fenêtre, et exécuter le programme sans utiliser le bouton **Run**.

> Dans le terminal, lancez la commande `rm Main.class` (pour supprimer ce fichier `Main.class`), puis la commande `ls` affichant le contenu du dossier.

Vous devriez constater que le dossier contient uniquement le fichier `Main.java`.

> Toujours dans le terminal, lancez la commande de compilation `javac Main.java`.

Cette commande a créé un fichier compilé `Main.class`, vous pouvez le vérifiez avec la commande `ls`.

> Lancer dans le terminal la commande d'exécution `java Main`.

Le programme s'est exécuté et le répertoire est inchangé (pas de nouveau fichier).

Au final, votre terminal doit ressembler à ceci :

```
> rm Main.class
> ls
Main.java
> javac Main.java
> ls
Main.class  Main.java
> java Main
Hello world!
> ls
Main.class  Main.java
```

Vous pouvez faire exactement la même manipulation sur votre ordinateur personnel (sous Linux. Pour Windows, les commandes doivent être adaptées), du moment que `java` a été correctement installé.

2 Gestion d'un panier pour un site de commerce

Nous souhaitons modéliser la tâche de construire un panier d'items qu'un utilisateur souhaite acheter, avec la possibilité d'ajouter des items, de modifier la quantité achetée, de calculer le coût total du panier, ...

On se place dans le scénario suivant. L'utilisateur met dans son panier 4 livres (`"Code Complete"` à 25 euros, `"Effective Java"` à 32 euros, deux copies de `"Clean Code"` à 29.50 euros).

1. Supprimer les instructions du `main`

(ou plutôt, mettez-les en commentaires, en ajoutant `//` en début de ligne. Pour toute la suite du sujet, lorsque vous devez supprimer des instructions du `main`, mettez-les en commentaires. Vous pouvez utiliser le raccourci clavier `Ctrl + k` puis `Ctrl + c` pour mettre une ou plusieurs lignes sélectionnées en commentaire).

2. Nous allons représenter toutes les données dans le programme.

   Créer plusieurs variables contenant les noms et prix de chaque livre et les quantités voulues de chaque, et une variable contenant le coût total du panier.

La syntaxe des instructions est disponible au lien suivant : `https://pageperso.lis-lab.fr/~arnaud.labourel/programmation1_2022/doc/doc_syntaxe.pdf`

3. Ajouter une instruction pour afficher en console le coût du panier, sous le format `Le coût du panier est xxx euros`.

Astuce 1 : pour taper `System.out.println` sous IntelliJ, il suffit de taper `sout` puis entrée.

Astuce 2 : `System.out.println` peut afficher plusieurs valeurs les unes à la suite des autres, il suffit de les séparer par des symboles `+`. Ces valeurs peuvent être des chaînes de caractères mais aussi des nombres. Par exemple on peut écrire :

```
System.out.println("sin(pi/3) = " + Math.sin(Math.PI / 3));
// ce qui affiche : sin(pi/3) = 0.8660254037844386
```

4. Rajouter des instructions pour afficher le contenu du panier, sous la forme :

```
Code Complete, 1 exemplaire(s) à 25.0 euro l'unité, total : 25.0 euros
Effective Java, 1 exemplaire(s) à 25.0 euro l'unité, total : 32.0 euros
Clean Code, 2 exemplaire(s) à 29.5 euro l'unité, total : 59.0 euros
Le coût du panier est 116.0 euros
```

5. Vous avez sûrement remarqué que vous avez écrit plusieurs fois les mêmes instructions, avec seulement de petites variations. Lorsqu'on programme, on essaie d'éviter de se répéter, c'est le principe DRY (*Don't Repeat Yourself*). Nous allons réorganiser le programme de sorte qu'il ne soit pas nécessaire de se répéter.

La première remarque est que les items sont tous constitués de trois informations : titre du livre, prix du livre, et quantité d'exemplaires dans la commande. Plutôt que de manipuler les trois informations séparément, nous allons les grouper pour les mettre dans une seule variable. Un tel groupement va constituer un *objet*. Nous aurons donc trois objets, un par livre.

Pour créer un objet, il faut un plan, le plan des objets livres. Les plans sont appelés *classes*, nous devons donc créer une classe des livres.

> Créer une classe `Book` en cliquant sur l'icône `"Add file"` dans la colonne "Files" et lui donner le nom `Book.java` (bien respecter la majuscule !).

6. Le fichier créé est initialement vide. On commence par écrire sur la première ligne la déclaration de classe `class Book` suivie d'une accolade ouvrante. `repl.it` complète automatiquement par une accolade fermante. C'est entre ces accolades que nous allons définir le contenu de la classe, par des déclarations.

Les déclarations sont de trois sortes : de *attribut*, de *méthode* et de *constructeur*. Une attribut est une information que possède chaque objet créé selon le plan de cette classe. Dans notre cas, les informations sont le titre, le prix, et le nombre d'exemplaires. Les objets livres pourront ne pas avoir les mêmes valeurs pour ces informations, mais ils doivent avoir ces trois informations.

Pour déclarer une attribut, dans le bloc de la classe, on ajoute une déclaration avec la syntaxe *identifiant du type de l'attribut* suivi de *identifiant de l'attribut*, terminé par un point-virgule. Par exemple, pour déclarer le titre on écrit :

```
String title;
```

> Ajouter la déclaration de titre, puis de la même façon, déclarer un prix unitaire et un nombre d'exemplaires pour les objets de la classe `Book`. Choisissez le type et le nom les plus appropriés dans chaque cas !

> 7. Replacez-vous dans la classe `Main`. Au début de la méthode `main`, déclarer trois variables de type `Book`, une par livre.

En effet, le fait de créer une classe a aussi créé le type des objets construits selon cette classe.

8. Pour l'instant, ces trois variables sont vides. Pour créer un objet selon la classe `Book`, on utilise le mot réservé de Java `new`, suivi du nom de la classe, suivi d'une paire de parenthèse. Par exemple :

```
Book codeComplete = new Book();
```

De même, initialiser les trois variables de type `Book`, avec trois objets différents de la classe `Book`.

9. Les attributs d'un objet s'utilisent comme des variables : ce sont des emplacements de mémoires dans lesquels on peut stocker des valeurs (entiers, nombres flottants, références d'objet). Pour accéder à une attribut, on utilise la syntaxe *identifiant de l'objet* suivi d'un point suivi de l'*identifiant de attribut*. Par exemple dans la méthode `main`, `codeComplete.title` est l'attribut `title` de l'objet référencé par la variable `codeComplete`.

Essayer d'afficher le titre de `codeComplete`. Que constatez-vous ?

10. Utiliser des affectations pour attribuer à chaque livre son titre. Modifier les instructions d'affichage pour qu'elles utilisent l'attribut `title` de chaque livre, plutôt que les variables de type `String` définies dans le `main` auparavant.

11. De la même façon, initialiser les attributs de prix et de nombre d'exemplaires de chaque objet.

12. Remplacer aussi le calcul du coût du panier en utilisant les attributs des livres directement.

13. Simplifier la méthode `main`, de sorte qu'il ne reste que les trois variables de type `Book`, la variable du prix du panier, les affectations et les instructions d'affichage du panier.

14. Simplifions le calcul de coût du panier. Pour chaque livre, il nous faut calculer le produit du prix du livre et du nombre d'exemplaires. Ce prix total ne dépend que des attributs du livre. Nous allons ajouter ce calcul dans la classe `Book` directement.

La deuxième sorte de déclaration pouvant être faite dans une classe est la déclaration de méthode. Une méthode d'un objet est un point d'accès permettant aux autres objets de faire des requêtes à cet objet. Par exemple, nous allons écrire une méthode `getTotalPrice` permettant de demander à un objet de type `Book` quel est le prix total des exemplaires commandés de ce livre.

Pour dire que les objets créés depuis la classe `Book` possèdent une méthode `getTotalPrice`, il faut déclarer la méthode dans la classe `Book`. Cette méthode calcule un prix, et donc lorsque la requête `getTotalPrice` est adressée à un objet de type `Book`, celui-ci doit répondre ce prix. On dit que `getTotalPrice` *retourne* le prix, qui est de type nombre à virgule (en java `double`).

La syntaxe d'une déclaration de méthode est *identifiant du type de retour* suivi de l'*identifiant de la méthode* suivi d'une paire de parenthèses (qui pourra contenir des paramètres), puis d'une paire d'accolades. La paire d'accolade définit un bloc qui doit contenir les instructions de la méthode.

```
double getTotalPrice() {
  // bloc d'instructions de la méthode
}
```

Les instructions de la méthode peuvent être arbitraires, mais la dernière instruction exécutées doit être une instruction `return`. Cette instruction permet de terminer le traitement de la méthode et de spécifier la valeur retournée. Sa syntaxe est spécifiée dans l'annexe.

Écrire la méthode `getTotalPrice` dans la classe `Book` de telle sorte qu'elle renvoie le produit du prix du livre par son nombre d'exemplaire.

15. On se replace dans la classe `Main`, au niveau du calcul du coût du panier. On peut maintenant remplacer chaque multiplication en utilisant la méthode que nous venons d'écrire. Pour obtenir le prix total pour les exemplaires d'un livre, il faut appeler la méthode `getTotalPrice` de ce livre. La syntaxe pour un appel de méthode est spécifiée dans l'annexe. Puisque la méthode va retourner le prix calculé, l'appel de méthode est une `expression` qui a comme valeur la valeur retournée. On peut donc directement écrire une addition dont chaque terme est un appel de méthode.

Réécrire le calcul du coût du panier avec trois appels de méthodes et des additions.

16. La plus grosse source de répétition de notre programme pour l'instant est l'affichage de chaque livre. La solution est de créer une méthode `display` dans la classe `Book`, qui se charge d'écrire la description du livre dans la console. Cette méthode effectue un affichage, mais n'a pas de valeur à répondre. On dit qu'elle ne retourne rien, et à la place de son type de retour, on utilise le mot réservé `void`. Aussi, on n'est pas obligé d'utiliser l'instruction `return` dans une méthode qui ne retourne rien.

Écrire la déclaration de la méthode `display` (laisser les instructions vides).

17. Pour les instructions, commencer par récupérer une des instructions d'affichage actuellement dans `main`, et la couper-coller vers `display`.

Tout ce qu'il reste à faire est de remplacer l'identifiant de l'objet de type `Book`, par l'identifiant de l'objet que nous voulons afficher.

Mais quel objet voulons-nous afficher ? La méthode `display` est une méthode déclarée dans la classe `Book`. Cela signifie que tout objet créé à partir de la classe `Book` (souvenez-vous qu'une classe est un plan de construction) possédera sa propre méthode `display`, comme il possède aussi ses propres attributs `title` ou `unitPrice`.

Pour que la méthode `display` d'un objet fonctionne correctement, il faut qu'elle utilise les attributs `title`, `unitPrice` et `quantity` (si vous les avez nommées ainsi) du même objet. Comme nous écrivons seulement un plan de construction, nous ne connaissons pas forcément tous les objets qui seront créés à partir de ce plan. Mais nous pouvons néanmoins parler de l'objet qui sera construit, en utilisant le mot réservé `this`. Ainsi `this.title`, utilisé dans la méthode `display`, fait référence à l'attribut `title` de l'objet dont la méthode `display` est évalué.

Corriger l'instruction d'affichage pour la rendre correcte.

18. Dans la classe `Main`, remplacer les instructions d'affichage des livres par des appels à la méthode `display`.

Vérifier que votre programme fonctionne toujours correctement.

19. Nous continuons de chasser les redondances. La façon dont nous initialisons les attributs des livres n'est pas satisfaisantes, puisque nous faisons trois fois les mêmes trois instructions. Par ailleurs, si je crée un quatrième livre, il se peut que j'oublie d'initialiser une des attributs de ce livre.

Pour améliorer cela, nous allons déporter les instructions d'initialisation directement dans la classe `Book`, en ajoutant un *constructeur*. Le rôle d'un constructeur est d'initialiser les attributs d'un objet (attention le terme est trompeur, le constructeur ne construit rien du tout).

Les constructeurs sont la troisième sorte de déclarations possibles dans une classe, après les attributs et les méthodes. Pour déclarer un constructeur, on utilise la syntaxe *identifiant de la classe* suivi d'une paire de parenthèses, suivi d'une paire d'accolades.

La paire de parenthèses peut contenir des paramètres séparés par des virgules, chaque paramètre se composant du type et de l'identifiant du paramètre. Les paramètres contiendront typiquement des valeurs à utiliser pour initialiser l'objet, par exemple les valeurs initiales des attributs de l'objet.

La paire d'accolades peut contenir des instructions, ce sont ces instructions qui serviront à initialiser l'objet.

> Créer une déclaration d'un constructeur dans la classe `Book`. Ce constructeur a 3 paramètres : `initTitle` de type `String`, `initPrice` de type `double` et `initQuantity` de type `int`. Pour l'instant, laisser le bloc d'instructions vide.

20. Regarder la classe `Main`. Des erreurs sont apparues. En effet, lorsqu'une classe définit au moins un constructeur, lors de la création d'un objet avec `new`, les arguments fournis dans l'instruction `new` doivent correspondre aux paramètres attendus par un des constructeurs.

Les arguments se placent entre les parenthèses qui suivent le nom de la classe, séparés par des virgules. Les arguments doivent respecter l'ordre et le type des paramètres du constructeur.

> Corriger les trois instructions `new` de la méthode `main`.

21. On retourne dans la classe `Book`, pour terminer le constructeur en écrivant ses instructions. Souvenez-vous que `this` dénote l'objet courant, et vous donne donc accès à ses méthodes et ses attributs.

> Écrire trois instructions, chacune initialisant une attribut du livre avec la valeur d'un des paramètres du constructeur.

Vérifier le bon comportement du programme.

22. On souhaite modifier le comportement de `display()` de sorte à ce qu'elle affiche `exemplaire` (sans s) s'il n'y a qu'un seul exemplaire du livre et `exemplaires` (avec s) s'il y a au moins deux exemplaires du livre.

> Modifier la méthode `display()` de `Book` afin d'obtenir l'affichage suivant :

```
Code Complete, 1 exemplaire à 25.0 euro l'unité, total : 25.0 euros
Effective Java, 1 exemplaire à 25.0 euro l'unité, total : 32.0 euros
Clean Code, 2 exemplaires à 29.5 euro l'unité, total : 59.0 euros
Le coût du panier est 116.0 euros
```

3 Gestion du panier

1. Imaginez que le panier contiennent 100 produits. Va-t-on devoir créer 100 variables, puis faire 100 instructions d'affichage ? Ce n'est pas raisonnable. Nous allons ajouter une nouvelle classe, dont les objets représentent les paniers des clients.

> Créer une classe `Cart`.

2. Quelles sont les attributs d'un panier ? Un panier est caractérisé par son contenu, c'est-à-dire la liste des livres contenus dans le panier.

Java propose de nombreuses classes pré-écrites et prêtes à être utilisée. Ces classes constituent la *librairie standard*. Parmi ces classes, certaines sont des classes représentant des listes. Pour créer un liste de livres, on écrit :

```
List<Book> listOfBooks = new ArrayList<>();
```

On reconnaît la syntaxe de la déclaration et de l'initialisation d'une variable d'identifiant `listOfBooks`. Son type, un peu particulier, est `List<Book>`, ce qui signifie *liste de livres*. L'initialisation est faite avec un nouvel objet (utilisation de `new`) construit avec selon le plan défini par la classe `ArrayList` (la présence des symboles `<>` sera expliquée plus tard dans le cours).

> Recopier cette déclaration dans la classe `Cart`, pour en faire une déclaration de attribut de cette classe.

Si vous exécutez le code, vous obtenez deux erreurs dont la première est

```
Cart.java:5: error: cannot find symbol
List<Book> listOfBooks = new ArrayList<>();
  symbol:   class List
  location: class Cart
```

Le compilateur ne trouve pas le terme `List` et il en est de même pour le terme `ArrayList` dans l'erreur qui suit. En effet, pour utiliser une classe de la librairie standard, il faut le déclarer par une déclaration d'import. Les classes à importer dans ce cas sont `java.util.List` et `java.util.ArrayList`.

> Sur la première ligne de la classe `Cart` (donc avant la déclaration de la classe), ajouter `import java.util.List;`. Sur la ligne d'après, procéder de même pour le terme `java.util.ArrayList`.

> 3. Ajouter une méthode `addBook` dans la classe `Cart`.

Cette méthode ne retourne rien. Elle prend en paramètre un objet de type `Book`. Pour mettre un paramètre à une méthode, faites comme pour les constructeurs : ajoutez-le entre les parenthèses de la déclaration de méthode, sous le format *identifiant du type* suivi de l'*identifiant du paramètre*.

4. Comme instruction de la méthode, on veut ajouter le livre à la liste constituant le panier. La liste `listOfBooks` est elle-même un objet, qui possède des méthodes. Ces méthodes permettent de faire des manipulations sur la liste.

Dans le bloc de la méthode `addBook`, commencer une instruction en tapant l'identifiant de la liste (`listOfBooks`) suivi d'un point.

Un menu déroulant s'affiche (parfois cela prend un peu de temps), listant toutes les méthodes de l'objet `listOfBooks`. L'une des méthodes s'appellent `add`, et comme son nom l'indique permet d'*ajouter* un élément à la liste. Les noms de méthodes sont choisis pour expliciter le rôle de la méthode.

Terminer l'instruction d'ajout du livre dans la liste, en fournissant à la méthode `add` son argument qui est le livre à ajouter.

5. Ajouter une méthode `display`, affichant tous les objets du panier.

Pour cela, il faut considérer tous les livres de la liste, un par un, et les afficher. Pour effectuer des instructions pour chaque élément d'une liste, on utilise une boucle `for`. La syntaxe de la boucle `for` en Java est :

```
for (Book oneBook : listOfBooks) {
  // instructions à faire pour chaque livre
}
```

Dans les parenthèses, on trouve d'abord l'*identifiant de type* des éléments de la liste, puis l'*identifiant de variable* qui sera utilisé pour chaque livre dans les instructions du bloc de la boucle (entre les accolades), puis le deux-points puis l'*identifiant de la liste*. Entre les accolades, se trouvent les instructions à effectuer sur chaque objet de la liste. Ces instructions peuvent utiliser l'identifiant `oneBook` pour faire référence à l'objet provenant de la liste.

Recopier la boucle, et mettre une instruction provoquant l'affichage du livre.

6. Dans la classe `Main`, créer une nouvelle variable `cart` de type `Cart`, initialisée avec un nouvel objet.

7. Utiliser la méthode `addBook` pour ajouter les trois livres dans le panier.

8. Remplacer l'affichage de chaque livre par un seul appel de méthode à l'objet `cart`.

9. Ajouter une méthode `getCost` à la classe `Cart`, permettant de calculer et retourner le coût du panier.

Pour calculer ce coût, il faut considérer tous les éléments du panier un par un, et ajouter leur prix totaux entre eux. On utilise une variable `sum`, initialement nulle, qui contiendra le total. Pour chaque livre, on ajoute son prix total à la variable `sum`.

10. Modifier la méthode `display` de la classe `Cart`, pour qu'elle affiche d'abord le coût total du panier, puis les livres.

Pour cela, elle doit calculer le coût total en utilisant sa propre méthode `getCost`, par un appel de méthode.

11. Modifier aussi la méthode `main` de la classe `Main`, pour que l'affichage reste le même qu'avant.