Feuille de TDs

1 CONCEPTS DE BASE

1. Les classes

On souhaite pouvoir gérer les ventes d'un magasin de musique. Quels sont selon vous les objets pertinents qui devront être manipulés ?

   a. Prenons comme premier exemple une classe **Instrument**. Ecrivez cette classe pour que chaque objet de cette classe contienne une **reference**, un **prix**, un **poids**.

   b. Quel sera le nom du fichier contenant cette classe ?

2. Instanciation
   a. Ecrivez une fonction main qui créé deux instruments de musiques.

3. Masquage
   a. On souhaite qu'il soit impossible de changer le prix d'un objet, comment faire ? Modifier le code en conséquence.
   b. Ecrivez une méthode permettant de modifier le prix, en contrôlant que le prix n'est pas négatif.
   c. Ecrivez une méthode permettant de modifier le prix, en contrôlant que le prix n'est pas négatif, et qu'il est modifié de maximum 10% par rapport au prix précédent.

2 MANIPULATION DE CHAINES DE CARACTERES

On souhaite afficher un code pour chaque instrument.

1. Créer une méthode dans votre classe **Instrument**. Cette méthode doit afficher la référence de l'objet.
2. Modifier votre méthode pour qu'elle affiche la référence si celle-ci fait moins de 5 caractères, sinon, afficher les 5 premiers caractères suivis de « … ». Pour cela, utiliser :
   - L'attribut length : qui renvoi le nombre de caractères d'un string
   - La fonction substring(int x, int y) : qui renvoi la partie du string comprise entre le xième et le yième caractère. Exemple :

     String toto = new String(« Bonjour ») ;
     System.out.println(toto.substring(0,3)) => affiche « bon »
   - La concatenation :
     - Soit avec la méthode s1.concat(String s2)
     - Soit avec la concaténation par « + »
3. Ecrire une méthode qui vérifie que la référence commence par « MUS ». Si ce n'est pas le cas, ajouter « MUS » au début de la référence.

Utiliser la méthode s1.compareTo(String s2), qui renvoi 0 si s1 et s2 ont la même valeur.

3 L'HERITAGE

On souhaite recenser les différents animaux au sein d'une zone de forêt. On définit trois catégories : les **Oiseaux**, les **Mammifères** et les **Poissons**.

1. Quelle serait la classe généralisant les trois classes ci-dessus ?
2. Donner des exemples de spécialisation de la classe **Mammifere**.
3. Dessinez la hiérarchie de classe.
4. Ecrivez la classe **Animal**. Un animal a un nom, un poids, et mange.
5. Ecrivez la classe **Mammifère**. Un mammifère est un animal qui a un nombre de pattes, et se déplace en marchant.
6. Ecrivez la classe **Rongeur**, qui est un mammifère qui grimpe et a un nombre de dents.
7. Sachant un Animal a, un Mammifère m, et un Rongeur r, déterminez si les phrases suivantes sont correctes :
    a) a.mange() ;
    b) m.mange() ;
    c) r.marche() ;
    d) r.grimpe() ;
    e) a.grimpe() ;
    f) r.nb_pattes = 4 ;
    g) m.nb_dents = 32 ;
8. Quelle phrase est correcte ?
    a) Animal a = new Rongeur() ;
    b) Rongeur r = new Animal() ;

4 POLYMORPHISME

On veut poser des pièges photos, un piège photo doit pouvoir prendre en photo un Mammifere ou un Rongeur.

1. Ecrire la classe **Piege_photo** contenant une méthode **photographier**.

2. On considère les classes d'instruments de musique suivants :
    - **Guitare**
    - **Basse**
    - **Flute**

    Un musicien doit pouvoir jouer de n'importe quel instrument. Comment faire ? Ecrivez la classe musicien et utiliser le polymorphisme pour une méthode jouer.

5 CONSTRUCTEURS

1. Qu'est-ce que le constructeur par défaut ?
2. Ecrire un constructeur initialisant les attributs à des valeurs par défaut pour la classe **Animal**.
3. Ecrire un constructeur prenant en argument les valeurs des attributs pour la classe **Animal.**
4. Ecrire un constructeur sur le même principe pour la classe **Mammifere.**
5. Ecrire le même constructeur en utilisant le mot clé **super**.
6. Ecrire une méthode main qui instancie un Mammifere « pecari », et un Rongeur « cabiai ». Quel est le problème de ce code ?
7. Ecrire un constructeur pour la classe **Rongeur** qui résout ce problème (toujours en utilisant le mot clé **super**).

Nous allons maintenant manipuler les références.

8. Commencez par créer un Mammifere m et un rongeur r.
9. Affectez ensuite à m la valeur de r. Quel objet contient m ? et r ? Qu'est devenu l'objet qui été contenu dans m ?
10. Modifiez le nom de m. Quel est le nom de r ?

6 ACCESSEURS

1. Réécrire la classe **Animal** pour éviter que l'on puisse modifier l'attribut poids, mise à part dans les classes filles.
2. Ecrire un getter permettant d'obtenir le poids s'il est différent de 0, et d'afficher un message d'erreur sinon.
3. Ecrire un setter permettant de modifier le poids, uniquement si la valeur est positive.
4. Ecrire une méthode main qui instancie un objet de la classe **Rongeur**, puis modifier son poids grâce au setter précédemment créé.

7 CLASSES ABSTRAITES

On souhaite que les animaux enregistrés contiennent des données précises. On veut donc éviter de pouvoir instancier des animaux ou des mammifères, uniquement des oiseaux, des poissons ou des rongeurs.

1. Réécrire l'entête de la classe **Animal** et de la classe **Mammifere**

On veut qu'une nouvelle classe **Biologiste**, puisse observer n'importe quel type d'Animal. Chaque Animal étant observé dans son milieu.

2. Ecrire la classe **Biologiste** contenant la méthode **observer**.
3. Réécrire la classe **Animal** en ajoutant la méthode abstraite **etreObserve**.

4. Réécrire les classe **Oiseau**, **Poisson** et **Rongeur**, ayant chacun une méthode **etreObserve** qui affiche un message adapté.

On implémente un système pour GPS permettant d'afficher différents éléments à l'écran. On a notamment :

- une carte affichant la position de l'utilisateur
- des boutons permettant d'envoyer des messages de secours à une personne de confiance lors de séjours en forêt. L'application contient pour l'instant deux boutons. Un bouton pour envoyer « à l'aide », un bouton pour envoyer « tout va bien ».

On souhaite que cette application puisse évoluer, pour pouvoir ajouter dans le futur d'autres widgets. On veut cependant s'assurer qu'un élément a toujours une taille, et quelle est positive.

5. Dessinez la hiérarchie de classe contenant les classes : **Element**, **Bouton**, **Bouton_aide**, **Bouton_ok**
6. Quelles sont les classes qui doivent être abstraites selon vous ?
7. Ecrire la classe **Element** avec une hauteur et une largeur, pour qu'on ne puisse pas mettre de taille négative (utiliser un setter et le masquage pour cela)
8. Ecrire la classe **Bouton** pour que tous les boutons aient une taille par défaut.
9. Ecrire la classe **Bouton_aide**, avec une méthode permettant d'augmenter sa taille de 1 cm en hauteur et en largeur. Puis une méthode permettant d'augmenter progressivement sa taille de n centimètres (n étant passé en paramètres).
10. Ecrire une fonction main, qui instancie un **Bouton_ok** et un **Bouton_aide**. Puis modifier le bouton de classe **Bouton_aide** pour qu'il soit plus gros de 10cm en hauteur et en largeur.

8 MEMBRES STATIQUES

Nous souhaitons pouvoir connaître le nombre d'animaux présents dans la base de données.

1. Ajouter à la classe **Animal** un attribut statique nb_animaux initialisé à 0. On note que cette variable appartient à la classe, elle a donc une valeur commune à tous les objets de cette classe.
2. On souhaite désormais pouvoir incrémenter cette variable lors de l'instanciation d'un objet de classe Animal.
    a. Commencez par écrire une méthode statique privée qui incrémente **nb_animaux**.
    b. Quelle méthode faut-il à présent modifier pour que la méthode précédente soit appelée à chaque instanciation ? Ecrivez ces modifications.
3. L'attribut étant privé, écrivez un getter permettant de récupérer la valeur de la variable.
4. Ecrivez une méthode main qui instancie deux animaux, puis affiche la valeur de la variable **nb_animaux**.

On souhaite coder une bibliothèque permettant d'effectuer quelques opérations sur des nombres.

5. Ecrire une classe **Math** ayant trois attributs statiques **pi**, **mem1** et **mem2**.
6. Ecrire des méthodes statiques qui permettent d'enregistrer des résultats de fonctions :
    a. Une méthode **save** qui enregistre dans **mem1** la valeur passée en paramètres, et enregistre dans **mem2** la valeur précédente de **mem1**.
    b. Deux getters qui retournent les valeurs de **mem1** et **mem2**.
7. Ajoutez à cette classe les méthodes statiques suivantes (qui doivent faire appel aux méthodes ci-dessus) :
    c. Une méthode calculant le périmètre d'un cercle de rayon r (r étant l'argument de la méthode)
    d. Une méthode calculant le reste de la division euclidienne de a par b (a et b étant les arguments de la méthode)
    e. Une méthode calculant le pgcd de a et b (a et b étant les arguments de la méthode). Utiliser pour cela la fonction précédente et l'algorithme d'euclide.

Algorithme d'euclide : à chaque itération, récuperer le reste r de la divison euclidienne de a par b. Puis a = b, et b = r, et on recommence. Réaliser cela jusqu'à obtenir un reste nul, le résultat est a.

8. Ecrivez une méthode main qui calcul le périmètre d'un cercle de rayon 12, le reste de la division euclienne de 7 par 3. Et le pgcd de 21 et 15. Afficher ensuite la valeur de **mem2**.

9 INTERFACE

On souhaite pouvoir réaliser un enregistrement de musique. On doit pour cela enregistrer la voix d'un **Chanteur** et les sons des **Instruments**.

1. Ecrire une interface **Enregistrable**, avec une méthode **enregistrer**.
2. Ecrire la classe **Chanteur** qui implémente l'interface **Enregistrable**.
3. Réécrire l'entête de la classe **Instrument** pour que cette classe soit enregistrable.
4. Ecrire une classe **Enregistreur** ayant une méthode **creerMusique** qui permet d'enregistrer trois sources de sons.

Notre classe **Biologiste** veut désormais protéger des espaces. Pour cela, elle souhaite pouvoir protéger tout élément de cette espace de manière adaptée.

5. Ecrire une interface **A_proteger,** ayant deux méthodes : **sauvegarder**, **deplacer**
6. Réécrire la classe **Mammifere** pour implémenter cette interface. Sauvegarder un mammifère consiste à interdire sa chasse, le déplacer à le capturer.
7. Ecrire une classe **Arbre**, qui implémente l'interface **A_proteger**. Sauvegarder un arbre consiste à interdire sa coupe, et le déplacer à le déraciner.
8. Réécrire la classe **Biologiste** pour qu'elle puisse protéger n'importe quel élément à protéger.

On souhaite améliorer nos boutons de GPS. Pour cela, on veut des boutons qui puissent être déplacés, et s'assurer qu'un bouton est toujours cliquable.

9. Ecrire une interface **Cliquable**, avec une méthode **clique**.

10. Ecrire une interface **Movable**, avec une méthode **move.**
11. Réécrire l'entête de la classe bouton. Définissez la méthode **clique,** un bouton doit afficher un message « bip » lorsqu'il est cliqué.
12. Ecrire une classe **Bouton_perso**, qui est un bouton déplacable contenant un message_perso.

10 TABLEAUX

On souhaite pouvoir ajouter et supprimer autant de boutons que souhaité sur notre GPS, et que ceux-ci s'affichent correctement.

1. La première solution est de considérer que l'écran du GPS contient 3 lignes et trois colonnes (9 cases). On peut donc stocker les boutons à l'aide d'un tableau statique. Ecrire une classe **Screen** contenant un tableau stockant des boutons. Pour empêcher les manipulations hasardeuses, on rendra se tableau privé, et on y accédera grâce à un setter permettant d'ajouter un bouton à la suite des autres.
2. Ecrire une fonction main qui instancie un **Screen**, et y ajoute un **Bouton_ok** et un **Bouton_aide**

Le problème d'une telle implémentation, c'est qu'elle limite l'utilisateur à neuf boutons. De plus, déplacer ou agrandir les boutons risque de poser problème. Essayons avec un tableau dynamique

3. Réécrire la classe **Bouton** pour qu'elle intègre une position.
4. Ecrire la classe **Screen_dyn**, qui contient un tableau dynamique de boutons.
5. Pour que la classe soit fonctionnelle, nous allons également rendre ce tableau **private**, et utiliser des accesseurs. Ecrivez une méthode qui parcourt les boutons existants et trouve le bouton ayant la position la plus éloignée de l'origine (origine = 0,0), vous pouvez pour cela créer des méthodes intermédiaires. Cette méthode doit ensuite ajouter un bouton (passé en argument) au tableau en le positionnant 2cm de plus en abscisse et en ordonné.
6. Ecrivez une méthode main qui instancie un **Screen_dyn** et y ajoute un **Bouton_aide**.

11 REDEFINITION DE METHODE

Nous avons créé un setter pour la classe **Animal** qui vérifie que le poids donné en entrée est bien positif. Nous souhaitons affiner cette fonction pour les rongeurs, car ceux-ci ne peuvent avoir un poids supérieur à 80 kilos (un capybara, le plus gros rongeur du monde, pèse entre 35 et 66 kilos).

1. Surchargez la méthode **setPoids** pour vérifier que le poids passé en arguments est entre 0 et 80.

On souhaite désormais améliorer la classe **Enregistreur**. On veut que la fonction **creerMusique** puisse prendre en arguments n'importe quelle composition d'objets **Enregistrable**.

2. Surchargez cette méthode pour qu'elle prenne en argument un tableau dynamique, et enregistre chaque objet du tableau.

Notre classe **Bouton_perso** ne fait qu'émettre un bip pour l'instant. On souhait qu'elle affiche également en suivant son message_perso.

3. Surchargez la méthode **clique**, appeler la fonction de la classe mère avec le mot clé super, puis afficher le message_perso.

Tout objet possède une méthode **toString**. Cette méthode ne prend aucun argument et renvoi toujours un String. C'est cette méthode qui est systématiquement appelée lorsqu'on convertit implicitement un objet en String. Notamment lorsqu'on cherche à afficher les informations de l'objet avec la méthode System.out.println(). En règle général, lors de la création d'une classe, on surcharge la méthode toString() pour afficher proprement l'objet.

4. Ajouter à votre classe Instrument la surchage de la méthode **toString()**. Cette méthode renverra un String de la forme :
   « Ref : valeur Ref | Poids : valeurPoids | Prix : valeurPrix »

12 FINAL

1. On souhaite que la classe **Bouton_aide** ne puisse être dérivée. Réécrivez son entête.
2. On souhaite que la méthode pgcd de la classe **Math** ne puisse être surchargée, réécrivez l'entête de la méthode.
3. On souhaite que la valeur de l'attribut **pi** de la classe **Math** ne puisse être modifiée, réécrivez la déclaration de cet attribut.

13 PROGRAMMATION GENERIQUE

Nous allons généraliser notre classe **Screen_dyn**. Nous allons lui permettre d'afficher n'importe quel type d'élément, du moment que celui-ci est cliquable.

5. Réécrivez la classe **Screen_dyn** en la rendant générique. Cette classe devra pouvoir stocker via un tableau dynamique des objets cliquables.

Nous allons maintenant étudier la classe **Map** existante en java. Cette classe représente un dictionnaire.

6. Créer une classe **Position** permettant d'enregistrer deux valeurs (une pour l'abscisse et une pour l'ordonnée)
7. Améliorer votre classe Screen pour que celle-ci utilise un objet de classe **Map** plutôt qu'un tableau dynamique classique. Celui-ci devra permettre d'associer une position à un élément cliquable (toujours de manière générique).

On considère qu'un **Screen_dyn** ne doit contenir qu'un seul **Bouton_aide**, et que celui-ci est toujours affiché en dernier.

8. Ecrivez une methode qui vérifie qu'il existe un **Bouton_aide** dans la **Map**. Cette méthode renvoi la position du Bouton_aide, et une Position null sinon.

9. Ecrivez une méthode qui permet d'ajouter un **Bouton** au Screen. Cette méthode doit d'abord vérifier si le bouton à ajouter est un Bouton_aide.
    a. Si c'est le cas, elle vérifie s'il existe déjà un bouton de cette classe dans la Map.
        i. Si oui, le nouveau bouton n'est pas ajouté.
        ii. Sinon, le bouton_aide est ajouté à la position 0,0 ;
    b. Sinon, la methode doit placer le nouveau bouton à la position du Bouton_aide, puis déplacer le Bouton_aide de 5cm en abscisse et en ordonnée.

Nous allons maintenant modéliser des commandes d'instruments de musique. La classe **Commande** permettra de stocker dans un tableau dynamique une liste d'instruments.

10. Ecrire la classe **Commande**

Nous souhaitons maintenant que cette liste puisse être affichée par ordre de prix croissant. Pour cela nous allons rendre la classe **Instrument** « comparable ».

11. Ecrire la classe **Instrument** pour implémenter l'interface **Comparable**.
12. Ajouter à cette classe un constructeur qui initialise l'instrument avec la référence et le poids passé en argument, et génère un prix aléatoirement.

    Pour cela, vous pouvez utiliser un objet de la classe **Random**, avec la méthode nextFloat(), qui génère un chiffre aléatoire entre 0 et 1.

13. Définissez la méthode monInstru.**compareTo**(Instrument i), qui renverra :
    c. 1 si le prix de monInstru est plus grand que le prix de l'instrument i.
    d. 0 si les prix sont égaux
    e. -1 sinon

Pour pouvoir vérifier que le tri a bien fonctionner, on souhaite afficher proprement les instruments de la commande.

14. Ajouter à votre classe **Commande** une surcharge de la méthode **toString()** qui affiche les instruments de son tableau.
15. Ecrire une méthode main qui :
    f. créé une commande
    g. ajoute trois objets à cette commande : une flute, une guitare, une basse
    h. trie les objets selon leur prix
    i. affiche la commande ainsi triée

14 INTROSPECTION

1. Ajouter à la classe **Commande** une méthode permettant de vérifier si une guitare est présente dans la commande.
2. Ajouter également une méthode qui parcours le tableau de la commande et affiche pour chaque élément sa classe :

« Ma classe : nomClasse»

15 LECTURE DE FICHIER + EXCEPTION

1. Ecrire une méthode qui ouvre un fichier (nom passé en paramètres) et affiche les lignes commençant par « % ».

Une manière classique de lire un fichier est d'utiliser deux classes :

- Un FileReader, créé avec un constructeur prenant en argument le nom du fichier que vous voulez manipulez. (Attention, dans eclipse, ce fichier doit être à la racine de votre projet : par défaut, votre projet se trouve dans C://Utilisateurs/nomUtilisateur/eclipse-workspace/nomProjet, en remplaçant nomUtilisateur par votre nom de session Windows, et nomProjet par le nom de votre projet eclipse).
- Un BufferedReader, créé avec un constructeur prenant en argument un FileReader.

Ensuite, il vous suffit d'utiliser depuis le BufferedReader la méthode readline() qui renvoi le fichier ligne après ligne sous forme d'un String.

ATTENTION : n'oubliez pas de fermer proprement le fichier, en appelant close() depuis le BufferedReader.

Une manière classique d'écrire dans un fichier est d'utiliser un FileOutputStream, créé avec un constructeur prenant en argument le nom du fichier dans lequel vous voulez écrire. Si le fichier n'existe pas, il sera créé à la racine du projet.

- Il vous suffit ensuite d'utiliser la méthode write() depuis ce FileOutputStream, qui prend en entrée un tableau de bytes.
- Pour obtenir un tableau de bytes à partir d'un String, vous pouvez appeler la méthode getBytes() depuis ce String.
- ATTENTION : n'oubliez pas de fermer proprement le fichier, en appelant close() depuis le FileOutputStream.

2. Ecrire une méthode qui créé une copie d'un fichier en remplaçant toutes les occurrences du mot « Fourmi » par « Velociraptor » dans un fichier.

Astuce : le plus simple est de récuperer l'intégralité du fichier dans un String, puis d'utiliser la méthode replace(String a, String b), qui remplace les occurrences de a par b.

3. Ecrire une classe **Remplaceur** qui a un attribut nom_de_fichier. Cette classe a une méthode proceed, qui copie un fichier en remplaçant toutes les occurrences d'un mot par un autre. (mot1 et mot2 seront passés en arguments)
4. Ajouter à la classe une méthode proceed (surcharge) qui supprime les occurrences du mot passé en paramètres.

Considerant la classe suivante :

```
class Except extends      Exception{
```

```
	public int	n;
	public Except(int n){
		this.n =	n;
	}

}
```

Etudier la fonction main qui suit :

```
public class Chemin{
	public static void	main	(String	args[])	{
		int	n	;
		Scanner clavier = new Scanner(System.in);
		System.out.print ("donnez un entier :") ;
		n	=	clavier.nextInt();

		try{
			System.out.println ("debut du premier bloc try");
			if(n!=0) throw	new	Except(n)	;
			System.out.println ("fin du premier bloc try");
		}
		catch(Except e){
			System.out.println ("catch 1 = "+e.n);
		}

		System.out.println ("suite du programme");

		try{
			System.out.println ("debut du second bloc try");
			if	(n!=1) throw new	Except(n)	;
			System.out.println ("fin du second bloc try");
		}
		catch(Except e)	{
			System.out.println ("catch 2 = "+e.n);
			System.exit(-1)	;
		}
		System.out.println ("fin programme");
	}
}
```

1. Que se passera-t ’il si la valeur tapée au clavier est 0 ?
2. Et si la valeur est 1 ?