Les TP ci-après ne sont que des propositions (à caractère indicatif et non impératif).
Il ne faut pas hésiter à tester tout un tas de variantes (selon ses préférences personnelles).

NB: chaque nouvelle classe développée dans un TP devra être placée
dans un **package** adéquat (dont le nom est à choisir) .

1. TP1 (prise en main du jdk)

Objectif : Edition, Compilation et Exécution sans eclipse , avec un simple éditeur de texte et le jdk :

A faire : Hello World !!!

2. TP2 (première classe simple, conventions JavaBean)

Objectif : Ecrire une classe Java dans les règles de l'art

A faire:

- Créer un nouveau projet java (basé sur la technologie "**maven**") de nom "***tpInit***" oiubien "***basesJava***") sous eclipse ou bien intelliJ
- Créer la classe "**MyApp**" ou "**ConsoleApp**" avec une méthode **main**() qui servira aux tests.
- Créer (et tester) une première version d'une classe "**Personne**" avec des attributs "**nom**", "**age**", "**poids**" déclarés provisoirement "public".
- Coder une méthode "public void **afficher**()" qui affiche à l'écran les valeurs des attributs. (+ tests)
- Rendre "**private**" les **attributs** de la classe "Personne"
- Générer les méthodes **get**Xxx()/**set**Xxx(...) (+tests)
- Coder quelques **constructeurs**. (+tests)
- Coder la méthode classique "public String **toString**()" (provenant de la classe Object) en y construisant une chaîne de caractères complète regroupant tous les attributs + return
- Reprogrammer la méthode afficher() de façon à ce quelle appelle toString() en interne.
- Créer deux instances p1 et p2 de la classe Personne avec les mêmes valeurs internes.
- Comparer ces 2 instances et afficher si (oui ou non) les valeurs internes sont identiques.
- Reprogrammer la méthodes "public boolean **equals**(Object obj)" sur la classe Personne. Cette méthode doit renvoyer "true" si et seulement si toutes les valeurs internes de this et de obj sont identiques.

Suite du TP :

Coder également une classe "**Bagage**" (avec label, poids, volume) . Avec des poids en grammes (Integer) et des volumes en litres (Double) avec constructeurs,.toString() , getter/setter , …
Tester le bon comportement de la classe Bagage via une nouvelle méthode testerBagage() appelée par la méthode main() .

Encore une fois la même chose !!! J'en ai mare de programmer des "get/set" pas passionnant disait le développeur "Jean Aimare" !!!
Heureusement, lombok est là pour vous simplifier la vie .
⇒ ajouter une dépendance lombok dans le pom.xml du projet

```
<dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.30</version>  < !-- ou bien version plus récente -->
</dependency>
```

=> utiliser @Getter() @Setter() @ToString() @NoArgsConstructor() …. au dessus de la classe Bagage et commenter tous les équivalents générés par l'IDE eclipse ou intelliJ .

⇒ lancer un build maven via "mvn clean package" pour vérifier un bon fonctionnement maven.
=> les getters/setters générés par maven lors de la compilation sont normalement bien reconnus par l'IDE intelliJ qui propose d'installer automatiquement le plugin pour lombok.
⇒ avec l'IDE eclipse il faut effectuer une installation manuelle du plugin via le menu **Help / install new software** , sélectionner le chemin "**https://projectlombok.org/p2** " puis se laisser guider puis redémarrer eclipse , puis menu "***project clean***" .

3. TP3 (Tableaux, String, ...)

testTableauMoyenne() :
créer un tableau de 6 nombres réels
et calculer la moyenne

Suite faculative du Tp :
Trouver et afficher la plus grande des valeurs du tableau:

testString() :
String s1 = "2023-01-17";
//extraire la partie mois de différentes façons et afficher cette valeur

String chaine="YTREZA" ;
//créer une nouvelle chaine inverse où tous les caractères sont dans l'ordre inverse
//afficher la chaine inverse

4. TP4 static – constante , ...

- Déclarer une constante Personne***.AGE_MAJORITE avec une valeur de 18***
- Ajouter les éléments suivants sur la classe "Personne" :
  * variable de classe privée "esperanceVie"
  * méthodes de classe getEsperanceVie () et setEsperanceVie()
  * méthode ordinaire estMajeur() retournant un booleen
- Déclencher le calcul racine carré de 81 dans la méthode principale main().

5. TP5 (classe "Employe" héritant de "Personne")

- Créer et tester une sous classe "**Employe**" (héritant de "Personne") et comportant un **salaire** en plus.
- Redéfinir la méthode **toString**() sur la classe "Employe" en y effectuant un appel au **toString**() de la classe "Personne" et en concaténant le salaire en plus.
- Bien soigner l'écriture des différents constructeurs .
- Effectuer quelques tests/essais au sein de MyApp.testerEmploye()

- Créer une classe Avion comportant un **nom** et une collection "personnes" de **Personnes.**
- Ajouter une méthode **addPersonne(Personne p)** dans la classe **Avion**
- Ajouter une méthode **initialiser()** dans la classe Avion de façon à créer et ajouter quelques Employés et Personnes dans la collection interne.
  NB : la méthode initialiser pourra par exemple appleler addPersonne(new Employe(pilote ou hotesse)) et addPersonne(new Personne("passagerClandestin")).
- Vérifier le **polymorphisme** s'effectuant automatiquement en codant le plus naturellement possible les méthodes afficher() / toString() au sein de la classe Avion .
- Tester le tout dans une méthode testerAvion() de MyApp .

6. TP6 (classe abstraite "ObjetVolant")

- Créer une nouvelle classe abstraite "**ObjetVolant**" comportant un attribut privé "couleur" de type String et les méthodes traditionnelles getCouleur() / setCouleur(...).
  Cette classe comportera une méthode abstraite **getPlafond**() prévue pour retourner l'altitude maximale que l'objet volant est capable d'atteindre .
- Retoucher la classe Avion de façon à ce quelle hérite maintenant de la classe abstraite ObjetVolant .

7. TP7 (interface "Descriptible" ou "Transportable")

- Créer une interface ***Transportable***" comportant les méthodes "**getDesignation**() et **getPoids**()"
- Remodeler la classe "Personne" de façon à ce qu'elle implémente les méthodes de l'interface "**Transportable**". [ex: getDesignation() peut appeler toString()]
- Restructurer la classe "**Bagage**" (avec label, poids, volume) de manière à ce qu'elle implémente l'interface "Transportable".
- Améliorer la classe Avion en y ajoutant une collection complémentaire "chosesTransportables" d'éléments "Transportable " (souvent dans la soute de l'avion).
- Modifier la méthode ".initialiser()" de la classe Avion en plaçant l'instance passager_clandestin (de la classe Personne) plutôt dans la collection .chosesTransportables que dans la collection .persoonnes
- La nouvelle version de la méthode afficher() de Avion pourra par exemple afficher les désignations de chacun des éléments et calculer la charge complète de l'avion (somme des poids des éléments).

8. TP8 (Exception):

Ecrire une toute petite application qui calculera la racine carrée du premier argument passé au programme.
Utiliser des traitements d'exception pour gérer les cas anormaux suivants:

- appel du programme sans argument ==> Array Index Out Of Bound Exception
- argument non numérique ==> Number Format Exception
- ...

V1 : simple try/catch dans main() sans if
V2: le main délègue le calcul à un objet de type "***SousCalcul.java***" (à programmer).
La méthode "***public double calculerRacine(double x)***" de SousCalcul devra tester si x est <0 et devra dans ce cas remonter une exception de type "***MyArithmeticException***" (à programmer en héritant de Exception ou RuntimeException et avec un constructeur de type:

MyArithmeticException(String msg) { super(msg); } .

NB (Sous eclipse): Après un premier lancement de l'application via "*click droit/Run as/java application*", un paramétrage de la partie "*Prog. arg*" de l'onglet "***arguments***" de "***Run/Run ...***" permet de préciser un (ou plusieurs) argument(s) de la ligne de commande/lancement [ex: 81 ou a9 ou rien]

Suite du Tp :
améliorer le code de la méthode .setAge() de la classe Personne en soulevant une exception appropriée via le mot clef throw en cas d'age négatif invalide et tester le tout au sein de MyApp.testerPersonne() .

9. TP9 (Collections & Generics)

Essais très progressifs sur des collections élémentaires (de "Integer" ou de "String"))

10. TP10 (Dates & ResourceBundle)

Insérer le fichier "**MyResources.properties**" dans le code source de l'application avec le contenu suivant:

msg.day=day

mas.month=month

msg.year=year

Développer ensuite une version française (_fr) avec "année" , "mois" et "jour".

Dans une sous méthode "*static void test_dates()*" appelée par main() effectuer les tâches suivantes:

- Récupérer les valeurs des messages "msg.day" , "msg.month" et "msg.year" .
- Récupérer les valeurs entières day , month, et year depuis la date d'aujourd'hui.
- Afficher proprement un message de type "annees: 2007, mois: 2 , jour: 12 " ou "year: 2007, month: 2 , day: 12 " via System.out.printf()

11. TP11 (Application *ou Applet* Dessin en awt/swing):

Ecrire une application ou un applet "Dessin" capable de dessiner des lignes, des rectangles ou des cercles avec une couleur que l'on choisira dans une liste déroulante.

phase1 --> générer la classe de la fenêtre principale (ou de l'applet et sa page html).
phase2 --> coder la structure graphique (imbrication de composants, layout)
phase3 --> coder les événements appropriés.

12. TP12 (Gestion des fichiers) :

Partie1 : Ecrire une application comportant (d'une façon ou d'une autre)
une instance d'une classe "JPanelPays" que l'on fabriquera.
La classe "JPanelPays" héritera de "javax.swing.JPanel" et comportera
une "JList" (imbriquée dans un JScrollPane") permettant d'afficher une
liste de pays que l'on récupèrera dans un fichier texte (c:\stage\
ressources\data_files\listePays.txt).
Conseil : remonter en mémoire les données dans un vecteur d'objets "Pays"
(nouvelle petite classe avec attributs "nomPays" et "capitale" et méthode toString()).
On pourra éventuellement développer un jeu consistant à trouver le pays
correspondant à une capitale sélectionnée aléatoirement.
Partie2 : Reprendre le Tp "ConsoleApp / AvionV2" et y ajouter du code permettant
de déclencher une sérialisation complète des données d'un "AvionV2".
--> rendre tout "Serializable" via des "implements" qui vont bien.
--> relire le fichier généré et remonter (via xxx.readObject()) les
données dans une seconde instance que l'on affichera à l'écran.

13. TP 13 (Accès aux bases de données) :

Ecrire (ou agrandir) une application comportant (d'une façon ou d'une autre)
une instance d'une classe "JPanelGeo" que l'on fabriquera.
La classe "JPanelGeo" héritera de "javax.swing.JPanel" et comportera
un "JTree" (imbriqué dans un JScrollPane") permettant d'afficher une
liste de régions et de départements que l'on récupèrera dans une
base de données (c:\stage\ressources\database\access\geo.mdb).
Conseil : remonter en mémoire les données dans des tableaux ou collections d'objets
"Departement" et "Region" (nouvelle petites classes avec attributs "nom" et "prefecture"
"..." et méthode toString()). Ces petits objets pourront également correspondrent à
la partie interne des noeuds de l'arbre (DefautMutableTreeNode du TreeModel).
On pourra éventuellement développer un jeu consistant à trouver le département
correspondant à une préfecture sélectionnée aléatoirement.

14. TP 14 (Gestion des threads) :

(dans un nouvel onglet "*OngletThread*")

- Partie 1 ==> démarrer en // 5 nouveaux threads qui afficheront "1" puis "2" puis ... puis

"20" dans une zone graphique réservée (propre à chacun des threads).

- Partie 2 ==> développer une classe "PoolDeRessource" permettant d'obtenir et de libérer des ressources (ex: Objet "String"pour simulation).
  Ce pool (que l'on limitera volontairement à 3 ressources) sera ensuite utilisé par les 5 threads qui afficheront le nom de la ressource obtenue et non plus la valeur d'un simple compteur.