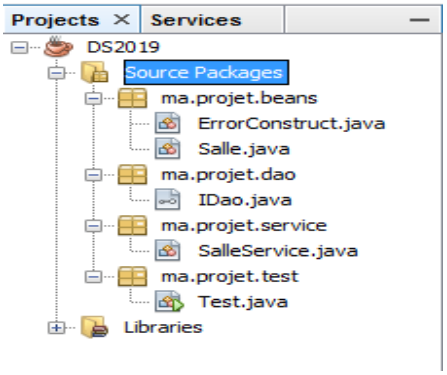
Correction DS 2019/2020

Exercice 1:

En tenant compte de la structure de projet suivante :



1) On vous demande de réaliser une application en JAVA permettant la gestion d'un ensemble de Salles. Une salle est définie par un identifiant (de type entier) qui est auto-incrémenté, une superficie ne dépassant pas les 100 m^2 et une description de son utilisation.
   a) Définir la classe Salle sachant qu'elle contient un constructeur permettant d'initialiser **tous** les attributs, une méthode d'affichage standard et les getters/setters permettant de modifier et de retourner les caractéristiques un objet.

```
package ma.projet.beans;
public class Salle {
   private int id;
   private int superficie;
   private String description;
    public Salle(int id,int superficie, String description) throws ErrorConstruct{
        this.id = id;
            if(superficie >100)throw new ErrorConstruct();
            else this.superficie = superficie;
        this.description = description;}
    public int getId() { return id;  }
     public int getSuperficie() {    return superficie; }
    public void setSuperficie(int superficie) {         this.superficie = superficie;            }
    public String getDescription() {return description ;         }
    public void setDescription(String description) {             this.description = description;         }
    public String toString() {                  return this.id + " " + this.superficie + " "+ this.description;      } }
```

   b) Implémenter l'exception liée à la superficie de la classe Salle.

```
package ma.projet.beans;

public class ErrorConstruct extends Exception {

   public ErrorConstruct(){       System.out.println("Erreur: la superficie ne doit pas dépasser 100");    } }
```

2) Définir une interface générique nommé **IDao** contenant les méthodes :

- ✓boolean create (T o) : Méthode permettant d'ajouter un objet o de type T.
- ✓boolean delete (T o) : Méthode permettant de supprimer un objet o de type T.
- ✓boolean update (T o) : Méthode permettant de modifier un objet o de type T.
- ✓T findById (int id) : Méthode permettant de renvoyer un objet dont id est passé en paramètre
- ✓List <T>findAll( ) : Méthode permettant de renvoyer la liste des objets de type T

```
package ma.projet.dao;
import java.util.List;
public interface IDao <T>{
   boolean create(T o);
 boolean update(T o);
 boolean delete(T o);
 List<T> findAll();
 T findById(int id); }
```

3) Définir la classe **SalleEnService** qui implémente l'interface **IDao<Salle>**. Cette classe permet de stocker un ensemble de salles dans une liste chainée.

```
package ma.projet.service;
import java.util.LinkedList;
import java.util.List;
import java.util.LinkedList;
import ma.projet.beans.Salle;
import ma.projet.dao.IDao;
public class SalleService implements IDao<Salle> {
  private List<Salle> salles;
 public SalleService() { salles = new LinkedList<Salle>();   }

    @Override
    public boolean create(Salle o) { return salles.add(o);}
    @Override
    public boolean update(Salle o) {
        for(Salle s : salles){
                if(s.getId() == o.getId()){
```

```
            s.setSuperficie(o.getSuperficie());
            s.setDescription(o.getDescription());
            return true;        } }
    return false;}


@Override
public boolean delete(Salle o) {        return salles.remove(o);    }
@Override
public List<Salle> findAll() {          return salles;      }
@Override
public Salle findById(int id) {
    for (Salle s : salles) {
            if (s.getId() == id)
                    return s; }
    return null;        }}
```

4) Tester les classes précédentes dans un programme principal. Celui-ci permet de :

- ✓ Créer trois salles. Simuler une exception dans le cas où la superficie dépasse le maximum (Définir un cas exceptionnel).
- ✓ Créer une instance de la classe SalleEnService , insérer les trois salles précédentes, et afficher le contenu de la liste, supprimer une salle puis modifier les informations d'une autre salle.

NB : n'utiliser que la méthode forEach() ou en cas de besoin des boucles for each

```
package ma.projet.test;
import java.util.Scanner;
import ma.projet.beans.ErrorConstruct;
import ma.projet.beans.Salle;
import ma.projet.service.SalleService;
public class Test {
   public static void main(String[] args) {
      SalleService ss = new SalleService();
      int comp = 1;
      try {   ss.create(new Salle(comp++,50, "Salle 1"));
        ss.create(new Salle(comp++, 25, "Salle 2"));
        ss.create(new Salle(comp++, 110,"Salle 3"));       }
```

```
catch (ErrorConstruct e)
{System.out.println("elle concerne la salle"+(comp-1));}
finally{ System.out.println("La liste des salles :");
        for (Salle s : ss.findAll())
                System.out.println("\t" + s);
        System.out.println("Supprimer la salle avec id = 1");
        ss.delete(ss.findById(1));
        System.out.println("Modifier la salle avec id = 2");
        Salle salle = ss.findById(2);
        System.out.println("\tSalle à modifier : " + salle);
        Scanner sc = new Scanner(System.in);
        System.out.println("Donner la nouvelle superficie :");
        salle.setSuperficie(sc.nextInt());
        System.out.println("Donner la nouvelle description :");
        salle.setDescription(sc.nextLine());
        ss.update(salle);

        System.out.println("La liste des salles après les mises à jour :");
        for (Salle s : ss.findAll())
                System.out.println("\t" + s);       }    } }
```

Exercice 2: Soit une interface fonctionnelle Exportable disposant d'une seule méthode void Exporter() et une classe Article dont le code Java est le suivant :

```
public class Article {

  protected int code;

  protected String nom;

  protected double prixHT;

  private static int comp;



   public Article(String nom,
double prixHT) {

this.code = ++comp;

this.nom = nom;

this.prixHT = prixHT; }
```

1 – Définir l'interface Exportable

```
public interface Exportable {

 void Exporter() ;  }
```

```
  public double prixTransport() {

returnprixHT * 0.05; } // 5% du Prix
Hors Taxes de l'article

public String toString() {

        return this.code + " " +
this.nom; }

public int getCode(){

    return this.code;}

}
```

2- Un article fragile est un article nécessitant un emballage (de type string) et dont le prix transport est deux fois le prix transport d'un article normal. Définir une classe Fragile. Redéfinir les méthodes qui vous semblent nécessaires pour le bon fonctionnement de la classe (ne pas définir les getters/setters).

```
public class Fragile extends Article {
   private String emballage;
   public Fragile(String emballage, String nom, double prixHT) {
      super(nom, prixHT);
      this.emballage = emballage;}
   public double prixTransport() {       return 2 * super.prixTransport();}
    public String toString() {        return super.toString()+" "+this.emballage; }}
```

3 - Un magasin sert à stocker un ensemble d'articles fragiles. Chacun de ces articles sera stocké dans un emplacement à identifiant unique (auto-incrémenté type Integer). Définir la classe Magasin sous forme d'un conteneur <Identifiant, Fragile>caractérisée par en plus :

- ✓ une méthode add() permettant d'ajouter un article fragile
- ✓ une méthode contains(Fragile f) qui vérifie si un article fragile est disponible au magasin. Utiliser une boucle for each.
- ✓ Une méthode qui permet d'afficher les articles fragiles du magasin.

```
import java.util.HashMap;
import java.util.Map;
public class Magasin {
    private Map<Integer,Fragile> fragiles;
    Integer cmp;
 public Magasin() {
       cmp = new Integer(0);
       fragiles = new HashMap<>();    }
```

```
public void add(Fragile a) {       fragiles.put(cmp++,a);  }
public boolean contains(Fragile a){
   for (Fragile fr : fragiles.values()) {
         if ( fr.getCode()== a.getCode()) {              return true;}
}return false;}
public void afficher(){    fragiles.values().forEach(System.out::println);}}
```

4 - Dans une classe principale de test :

- ✓ Ajouter deux articles fragiles dans un magasin.
- ✓ Définir un troisième article fragile, vérifier son existence dans le magasin sinon le rajouter.
- ✓ Afficher le contenu du magasin.
- ✓ Effectuer une exportation du deuxième article du magasin. Pour cela utiliser un objet d'une classe anonyme et une expression Lambda. L'exportation d'un article consiste à donner la main à l'utilisateur via son clavier, pour un article fragile donné et de préciser son pays de destination et les droits de douane en dinars (type en java : double).

```
import java.util.Scanner;
public class Principale {
      public static void main(String[] args) {
      Fragile fr1 = new Fragile("bouteille","eau",10);
      Fragile fr2 = new Fragile("bouteille","lait",20);
      Magasin mg = new Magasin();
      mg.add(fr1); mg.add(fr2);
      Fragile fr3 = new Fragile("cuir","veste",220);
      System.out.println(mg.contains(fr3));
      mg.afficher();

      Exportable exp = ()-> {
         System.out.println("l'article fragile à exporter:");
         System.out.println(fr2);
         Scanner sc = new Scanner(System.in);
         System.out.println("donner les droits de douane" );
         sc.nextDouble();
          System.out.println("donner le pays de destination" );
         sc.next();       };
      exp.Exporter();
   }
```

Annexe

java.util.Scanner	**Object**	**java.util.StringTokinezer**	**String**
Scanner(System.in); String next() String nextLine() booleannextBoolean() int nextInt() double nextDouble() Boolean hasNext() Boolean hasNextLine() booleanhasNextBoolean() booleanhasNextInt() booleanhasNextDouble()	String toString() booleanequals() finalize() clone()	StringTokenizer (String) StringTokenizer (String, String) StringTokenizer (String , String, boolean) String nextToken() booleanhasMoreTokens() int countTokens()	int length() int indexOf(char , int) String substring(int, int) boolean contains(String) boolean equals (String)
	Integer		
	intparseInt(String) : static		
Exception	**Class**	**StringBuffer**	**Math**
String getMessage()	String getName() Class getSuperclass() String toString() newInstance()	int length() int capacity() StringBufferappend(type de base) StringBufferinsert(int, type de base) StringBufferreverse()	PI: static double sqrt(double): static double random(): static
	java.util.Arrays(méthodes statiques)		

Optional <T> **OptionalInt**	int binarySearch(char[ ]) int binarySearch(int[ ]) int binarySearch(Object[ ]) sort(char[ ])	sort(int[ ]) sort(Object[ ]) sort(char[ ] a, int, int) fill(char[ ] , char) sort(Object[ ], Comparator)	fill(int[ ], long) fill(char[ ] , char, int , int) booleanequals(char[ ] , char[ ]), booleanequals(int[ ] , int[ ])
booleanisPresent() T get() Integer get(): pour OptionalInt			
Interface Entry<K,V>	**java.util.Collections**(méthodes statiques)		
K getKey(); V getValue(); setValue(V);	sort(List) sort(List, Comparator) shuffle (List) reverse(List)	fill (List, Object) copy(List, List) max (Collection, Comparator) min (Collection, Comparator)	binarySearch(List, Object) binarySearch(List, Object, Comparator)

Interface Collection<T> **java.util.Collection**	**Interface List <T>** **java.util.List**	**Interface Map <K,V>** **Java.util.Map**
int size() booleanisEmpty() boolean contains(T) add(T) remove(T) clear() T[] toArray() Iterator<T> iterator() forEach(Consumer<?superT>)) stream()	add(int, T) set(int,T) T get(int) remove(int) indexOf(T) lastIndexOf(T) subList(int,int) ListIterator<T>listIterator()	put(K, V) V get(K) String remove(K) boolean remove (K, V) booleancontainsKey(K) booleancontainsValue(V) int size() booleanisEmpty() putAll(Map) clear() Set<K>keySet(); Collection<V>values() Set<Entry<K,V>>entrySet() Entry: static
java.util.Iterator	**java.util.ListIterator**	
booleanhasNext() T next() remove()	booleanhasNext() T next() int previousIndex() ; remove() booleanhasPrevious() T previous()	