Java(MVC)

1.Objectifs

Dans un premier temps, la notion de type construit est introduite comme une classe possédant des attributs mais n'ayant pas de méthodes. Sur un exemple de mise en œuvre de fonctions (méthodes de classe), on se familiarisera avec l'appel de fonction (paramètres d'appel, valeur retournée) et enfin l'écriture de leurs réalisations. L'application à réaliser est une gestion de résultats d'élèves.

Dans un deuxième temps nous découvrirons plus pratiquement les concepts basiques de la programmation objet. Pour cela, nous aborderons les notions suivantes :

les classes et les services (méthodes)

la création d'objet (new)

la relation entre clients : l'envoi de message (un objet sollicite un service d'un objet connu)

les variables et méthodes de classe

2 Type construit

Nous désirons gérer les résultats d'élèves d'une promotion dans un tableau de **Resultats**. Un **Resultat** étant un type construit associant un nom d'élève (**String**) et une note (**int**).

Définissez la classe Resultats.

2.1 Utilisation de fonctions

La taille du tableau dépendra du nombre d'élèves demandé. Nous désirons posséder les fonctionnalités suivantes pour assurer la gestion des résultats :

créer un tableau de **Resultats** : fonction ***creation***,

saisir les résultats de chaque élève: fonction ***saisir***

, afficher les résultats de tous les élèves : fonction

afficher,

connaître la moyenne (float) de la promo : fonction

moyenne,

connaître la meilleure note (int) de la promo : fonction

meilleureNote

La fonction main suivante fait appel à ces fonctionnalités :

//fichier

UtilisationResultats.java

```
import java.util.*;
 public class UtilisationResultats {



      // les fonctions
   public static void main( String[]
                  args){

      Resultats[] resultatsMathematique;

      System.out.print("\n Creation du tableau de resultats en
      Mathematique \n"); resultatsMathematique =
      GestionResultats.creation();
      GestionResultats.saisir(resultatsMathematique);
      System.out.print("\n Moyenne Math : " +
      GestionResultats.moyenne(resultatsMathematique));
      System.out.print("\n Meilleure note Math : " +
      GestionResultats.meilleureNote(resultatsMathematique));
      System.out.print("\n Les résultats de Math : \n");
      GestionResultats.afficher(resultatsMathematique);
   }
}
```

.

Ecrivez les fonctions creation, saisir, afficher, moyenne, meilleureNote dans la classe GestionResultats. Vous utiliserez, pour les entrées au clavier, des méthodes d'instance de la classe Scanner.

Pour vous aider dans l'utilisation de la classe Scanner:

Construire un objet scan de type Scanner et qui lira sur l'entrée clavier : Scanner scan = new Scanner (System.in);

Deux méthodes d'instance de la classe Scanner: String

next()
Finds and returns the next complete token from this scanner.

int nextInt()
Scans the next token of the input as an int.

3 Découverte des objets: la gestion d'un stock

Il s'agit de développer et étendre un logiciel simple de gestion de stock. Celui-ci sera à terme constitué de trois classes : Produit, Stock et Producteur.

Un producteur fabrique des produits et les range dans un stock unique. A un moment donné le producteur ne connaît que le produit qu'il est en train de créer. Un stock contient de nombreux produits (qui pourraient être produits par plusieurs producteurs).

3.1 Réalisation de la classe Produit

3.1.1 Première version

Chaque Produit a un attribut name de type String qui est privé.

Le constructeur de Produit à pour paramètre le nom du Produit à construire. Les méthodes d'instance de la classe **Produit** sont :

- **getName()** qui renvoie le nom du **Produit** courant.
- **toString()** qui renvoie la chaîne de caractères qui décrit le

Produit courant.

3.1.2 Deuxième version : ajout d'un numéro au Produit

On souhaite en plus du nom du **Produit** associer aux différents produits un numéro unique : **number**. Pour cela, la classe mémorise le nombre d'instances qu'elle a créées et chaque nouveau **Produit** reçoit comme numéro **number** la valeur de ce compteur. La numérotation débutera à 1.

Un objet **Produit** pourra donner son numéro par la méthode d'instance **getNumber()**. Il sera possible de connaître le nombre total de produits créés par la méthode de classe **getNumberCreated()**. Il conviendra de modifier la méthode d'instance **toString()** pour qu'elle intègre le numéro et de couvrir les modifications dans le **main** de test.

Etendez la classe Produit pour intégrer ces modifications.

3.2 Réalisation de la classe Stock

La classe **Stock** gère un ensemble d'objets **Produit**. Le contenu d'un **Stock** est rangé dans un tableau. La taille maximale d'un **Stock** est fixée à sa création. Dans un premier temps, les méthodes ne gèrent pas les erreurs d'ajout d'un produit dans un stock plein ou de retrait d'un produit d'un stock vide.

Vous devez réaliser la classe Stock dont les différents éléments sont résumés ci-dessus : Chaque **Stock** a comme attribut :

size : le nombre de **Produit** actuellement dans le **Stock**.

content : un tableau de **Produit.**

Le constructeur de **Stock** a comme paramètre la taille maximale du **Stock** à construire. Les méthodes d'instance de la classe **Stock** sont :

add(Produit) : rajoute un nouveau produit dans le **Stock** courant.

remove() : retire le dernier produit ajouté au **Stock** courant et rend le **Produit** oté en résultat

isEmpty() : permet de savoir si le **Stock** courant est vide

isFull() : permet de savoir si le **Stock** courant plein

getSize() : permet de connaître le nombre de produits dans le **Stock** courant

toString() : rend une chaîne de caractères décrivant le **Stock** courant

Le **main** de la classe **Stock** servira uniquement à tester la classe.

3.2.1 Première version

Dans une première version, les méthodes ne gèrent pas les erreurs d'ajout d'un produit dans un stock plein ou de retrait d'un produit d'un stock vide.

Ecrivez une première version de la classe Stock.

3.2.2 Deuxième version : levée d'exceptions

Les méthodes **add** et **remove** de la classe **Stock** devront gérer les erreurs d'ajout d'un **Produit** dans un stock plein ou de retrait d'un **Produit** d'un stock vide, en levant des exceptions (**StockFull** ou **StockEmpty**) que vous aurez à définir.