Hello ,

Netsawer lazzem twasa3 Beliik chwya khater l corr moch bel order taa les Exs

Just you need to look for the name of your desired class and make sure that you will find it here inshallah. Good Luck!

Best,

Yahyaa

--------------------------------------

**public** **abstract** **class** Affichable {

**public** **abstract** **void** affiche();

**public** **static** **void** main(String[] args) {

Affichable[] objets = **new** Affichable[2];

objets[0] = **new** Entier(28);

objets[1] = **new** Flottan(3.27);

**for** (**int** i = 0; i < objets.length; i++) {

objets[i].affiche();

}

}

}

---------------------------------------------------------------

**public** **class** ClassA {

**public** **static** **int** f(**int** a) {**return** 2\*a;}

**public** **static** **int** g(**int** a) {**return** a+*f*(a/2);

}

}

-----------------------------------------------------------------

**public** **class** ClassB {

**public** **static** **int** f(**int** a) { **return** 3\*a; }

**public** **static** **int** g(**int** a) {

**return** ClassA.*f*(2\*a); }

**public** **static** **void** main(String[] args) {

System.***out***.println(*f*(2));

System.***out***.println(*g*(3));

System.***out***.println(ClassA.*f*(2));

System.***out***.println(ClassA.*g*(3));

} }

----------------------------------------------------------------------

**public** **class** Developpeur **extends** person {

**private** **double** salaire;

**public** Developpeur(String nom, String prenom, String mail, String telephone, **double** salaire) {

**super**(nom, prenom, mail, telephone, salaire);

**this**.salaire = salaire;

}

**public** **double** calculerSalaire() {

**return** salaire \* 1.20; // Augmentation de 20%

}

}

----------------------------------------------------------------------

**public** **class** Employe **extends** Personne {

**private** **static** **final** String ***SOCIETE*** = "MICRORDI";

**private** **float** salaire;

**private** **int** numero;

**private** **static** **int** *nombre* = 0;

// Constructeur par défaut

**public** Employe() {

**super**();

**this**.salaire = 0;

**this**.numero = ++*nombre*;

}

// Constructeur avec salaire et numéro

**public** Employe(**float** salaire, **int** numero) {

**super**();

**this**.salaire = salaire;

**this**.numero = numero;

*nombre*++;

}

// Constructeur avec personne, salaire et numéro

**public** Employe(Personne personne, **float** salaire, **int** numero) {

**super**(personne);

**this**.salaire = salaire;

**this**.numero = numero;

*nombre*++;

}

// Accesseurs et mutateurs

**public** **float** getSalaire() {

**return** salaire;

}

**public** **void** setSalaire(**float** salaire) {

**this**.salaire = salaire;

}

**public** **int** getNumero() {

**return** numero;

}

**public** **static** **int** getNombre() {

**return** *nombre*;

}

// Redéfinition de la méthode toString

@Override

**public** String toString() {

String personneString = **super**.toString();

**return** personneString + " et Employé à [SOCIETE=" + ***SOCIETE*** + ", salaire=" + salaire + ", numero=" + numero + "]";

}

**public** **static** **void** main(String[] args) {

}

}

-----------------------------------------------------------------------------

**public** **class** Entier **extends** Affichable {

**private** **int** entier;

**public** Entier(**int** entier) {

**this**.entier = entier;

}

**public** **void** affiche() {

System.***out***.println("Je suis un entier de valeur " + entier);

}

}

-----------------------------------------------------------------------------------------

**public** **class** Flottan **extends** Affichable {

**private** **double** flottant;

**public** Flottan(**double** flottant) {

**this**.flottant = flottant;

}

**public** **void** affiche() {

System.***out***.println("Je suis un flottant de valeur " + flottant);

}

}

---------------------------------------------------------------------------------------------

**public** **class** GroupeP {

**private** Polygone[] polygones;

**private** **int** compteur;

**public** GroupeP(**int** x) {

polygones = **new** Polygone[x];

compteur = 0;

}

**public** **boolean** ajouterPolygone(Polygone p) {

**if** (compteur < polygones.length) {

polygones[compteur++] = p;

**return** **true**;

}

**return** **false**;

}

**public** **double** perimetre() {

**double** totalPerimetre = 0;

**for** (**int** i = 0; i < compteur; i++) {

totalPerimetre += polygones[i].perimetre();

}

**return** totalPerimetre;

}

**public** String toString() {

StringBuilder sb = **new** StringBuilder();

**for** (**int** i = 0; i < compteur; i++) {

sb.append(polygones[i]).append("\n");

}

**return** sb.toString();

}

}

--------------------------------------------------------------------------------------------

**public** **class** manager **extends** person {

**private** **double** salaire;

**public** manager (String nom, String prenom, String mail, String telephone, **double** salaire) {

**super**(nom, prenom, mail, telephone, salaire);

**this**.salaire = salaire;

}

**public** **double** calculerSalaire() {

**return** salaire \* 1.35;

}

}

----------------------------------------------------------------------------------------

**public** **class** Modification {

**public** **static** **void** modif1(**int** x) { x+=2; }

**public** **static** **void** modif2(**int** x) { x=5; }

**public** **static** **void** main(String[] args) {

**int** x=24;

*modif1*(x);

System.***out***.println(x);

x=24;

*modif2*(x);

System.***out***.println(x);

} }

----------------------------------------------------------------------------------------------

**public** **abstract** **class** person {

**public** **static** **int** *compteur* = 0;

**protected** **int** id;

**protected** String nom;

**protected** String prenom;

**protected** String mail;

**protected** String telephone;

**protected** **double** salaire;

**public** person(String nom,String prenom,String mail,String telephone,**double** salaire) {

**this**.id = ++*compteur*;

**this**.nom = nom;

**this**.prenom = prenom;

**this**.mail = mail;

**this**.telephone = telephone;

**this**.salaire = salaire;

}

**public** person() {

**this**.nom = " ";

**this**.prenom = " ";

**this**.mail = " ";

**this**.telephone = " ";

**this**.id = 225844;

**this**.salaire = 1000;

}

**public** String toString() {

**return** **null**;

}

**public** **abstract** **double** calculerSalaire();

}

-----------------------------------------------------------------------------------------------

**public** **class** Personne {

**private** String nom;

**private** String prenom;

**private** **int** age;

**private** String genre;

// Constructeur par défaut

**public** Personne() {

**this**.nom = "";

**this**.prenom = "";

**this**.genre = "";

}

// Constructeur complet

**public** Personne(String nom, String prenom, **int** age, String genre) {

**this**.nom = nom;

**this**.prenom = prenom;

setAge(age);

**this**.genre = genre;

}

// Constructeur de copie

**public** Personne(Personne personne) {

**this**.nom = personne.nom;

**this**.prenom = personne.prenom;

**this**.age = personne.age;

**this**.genre = personne.genre;

}

// Accesseurs et mutateurs

**public** String getNom() {

**return** nom;

}

**public** **void** setNom(String nom) {

**this**.nom = nom;

}

**public** String getPrenom() {

**return** prenom;

}

**public** **void** setPrenom(String prenom) {

**this**.prenom = prenom;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**if** (age >= 0 && age <= 130) {

**this**.age = age;

} **else** {

System.***out***.println("L'âge doit être compris entre 0 et 130 ans.");

}

}

**public** String getGenre() {

**return** genre;

}

**public** **void** setGenre(String genre) {

**if**(genre.equals("Masculin") || genre.equals("Feminin")) {**this**.genre = genre;}

**else**{

System.***out***.println("La Personne doit etre Masculin ou Feminin.");

}

}

// Méthode pour vérifier si deux personnes ont le même nom de famille

**public** **void** sameLastName(Personne p) {

**if** (**this**.nom.equals(p.getNom())) {

System.***out***.println("Les deux personnes ont le même nom de famille.");

} **else** {

System.***out***.println("Les deux personnes n'ont pas le même nom de famille.");

}

}

// Méthode pour retourner la personne la plus âgée

**public** Personne oldest(Personne p) {

**if** (**this**.age > p.getAge()) {

**return** **this**;

} **else** {

**return** p;

}

}

// Méthode pour vérifier si la personne est majeure

**public** **boolean** estMajeure() {

**return** age >= 18;

}

// Redéfinition de la méthode toString

@Override

**public** String toString() {

String genreString = genre.equals("Masculin") ? "un homme" : "une femme";

String majeurString = estMajeure() ? "majeur" : "mineur";

**return** "Bonjour, je m'appelle " + prenom + " " + nom + ".\n" +

"Je suis " + genreString + ".\n" +

"Je suis " + majeurString + ".";

}

// Redéfinition de la méthode equals

@Override

**public** **boolean** equals(Object o) {

**if** (**this** == o) **return** **true**;

**if** (!(o **instanceof** Personne)) **return** **false**;

Personne personne = (Personne) o;

**return** age == personne.age &&

nom.equals(personne.nom) &&

prenom.equals(personne.prenom) &&

genre.equals(personne.genre);

}

**public** **static** **void** main(String[] args) {

// Création d'une première personne p1 par le constructeur par défaut

Personne p1 = **new** Personne();

// Modification des informations de la personne p1 avec les vôtres

p1.setNom("Chammami");

p1.setPrenom("Yahia");

p1.setAge(23);

p1.setGenre("Masculin");

// Affichage des informations de la personne p1

System.***out***.println("Informations de la personne p1 :");

System.***out***.println(p1);

System.***out***.println();

// Création d'une seconde personne p2 par clonage de p1

Personne p2 = **new** Personne(p1);

// Modification du nom et de l'âge de p2

p2.setNom("Ben karim");

p2.setAge(14);

// Affichage des informations de la personne p2

System.***out***.println("Informations de la personne p2 :");

System.***out***.println(p2);

System.***out***.println();

// Création d'une troisième personne p3 avec les arguments spécifiés

Personne p3;

p3 = **new** Personne("Dupond", "Jean", 131, "Masculin");

System.***out***.println("Nom de la personne p3 : " + p3.getNom());

System.***out***.println("Âge de la personne p3 : " + p3.getAge());

}

}

----------------------------------------------------------------------------------

**public** **class** Point2D {

// Variables d'instance privées pour les coordonnées x et y

**private** **double** x;

**private** **double** y;

// Constructeur par défaut initialisant le point à l'origine (0,0)

**public** Point2D() {

**this**.x = 0;

**this**.y = 0;

}

// Constructeur initialisant le point avec des coordonnées spécifiées

**public** Point2D(**double** x, **double** y) {

setX( **this**.x = x); /////

setY(**this**.y = y); /////

}

// Constructeur permettant de créer un point à partir d'un autre point

**public** Point2D(Point2D p) {

**this**.x = p.getX();

**this**.y = p.getY();

}

// Getter pour obtenir la coordonnée x du point

**public** **double** getX() {

**return** **this**.x;

}

// Getter pour obtenir la coordonnée y du point

**public** **double** getY() {

**return** **this**.y;

}

// Les Setter

**public** **void** setX(**double** x) {

**if** (x >= -50 && x <= 50) {

**this**.x = x;

} **else** {

System.***out***.println("La valeur de x doit être comprise entre -50 et 50.");

}

}

**public** **void** setY(**double** y) {

**if** (y >= -50 && y <= 50) {

**this**.y = y;

} **else** {

System.***out***.println("La valeur de y doit être comprise entre -50 et 50.");

}

}

// Méthode pour calculer la distance entre le point et l'origine (0,0)

**public** **double** distance() {

**return** Math.*sqrt*(**this**.x \* **this**.x + **this**.y \* **this**.y);

}

// Méthode pour calculer la distance entre le point et un autre point spécifié

**public** **double** distance(Point2D p) {

**double** dx = **this**.x - p.getX();

**double** dy = **this**.y - p.getY();

**return** Math.*sqrt*(dx \* dx + dy \* dy);

}

// Méthode pour effectuer la translation du point avec un vecteur (a, b)

**public** **void** translate(**double** a, **double** b) {

**this**.x += a;

**this**.y += b;

}

// Redéfinition de la méthode toString pour afficher les coordonnées du point

@Override

**public** String toString() {

**return** "(" + **this**.x + ", " + **this**.y + ")";

}

}

----------------------------------------------------------------------------------------------

**import** java.util.Arrays;

**public** **class** Polygone {

**static** **int** *comptpoly* = 0;

Point2D[] sommets;

**public** Polygone(Point2D[] sommets) {

**this**.sommets = sommets;

*comptpoly*++;

}

**public** Point2D getSommet(**int** i) {

**return** sommets[i];

}

**static** **double** longueur(Point2D p, Point2D q) {

**return** p.distance(q);

}

**double** perimetre() {

**double** peri = 0;

**int** n = sommets.length;

**for** (**int** i = 0; i < n; i++) {

peri += *longueur*(sommets[i], sommets[(i + 1) % n]);

}

**return** peri;

}

**public** String nom() {

**return** "Polygone";

}

**public** String toString() {

**return** Arrays.*toString*(sommets);

}

@Override

**public** **boolean** equals(Object that) {

**if** (**this** == that)

**return** **true**;

**if** (that == **null**)

**return** **false**;

**if** (**this**.getClass() != that.getClass())

**return** **false**;

Polygone other = (Polygone) that;

**return** Arrays.*equals*(**this**.sommets, other.sommets);

}

**public** **static** **void** main(String args[]) {

Point2D a = **new** Point2D(0, 0);

Point2D b = **new** Point2D(0, 3);

Point2D c = **new** Point2D(4, 0);

Triangle triangle = **new** Triangle(a, b, c);

System.***out***.println("rNom du triangle: " + triangle.nom());

System.***out***.println("Sommets du triangle: " + triangle);

System.***out***.println("Périmètre du triangle: " + triangle.perimetre());

// vérifier et afficher si le triangle est rectangle

**if** (triangle.estTriangleRectangle()) {

System.***out***.println("Le triangle est rectangle.");

} **else** {

System.***out***.println("Le triangle n'est pas rectangle.");

}

Point2D d = **new** Point2D(4, 3);

Quadrilatere quadrilatere = **new** Quadrilatere(a, b, c, d);

System.***out***.println("Nom du quadrilatère: " + quadrilatere.nom());

System.***out***.println("Sommets du quadrilatère: " + quadrilatere);

System.***out***.println("Périmètre du quadrilatère: " + quadrilatere.perimetre());

// vérifier et afficher si le quadrilatère est un parallélogramme

**if** (quadrilatere.estParallelogramme()) {

System.***out***.println("Le quadrilatère est un parallélogramme.");

} **else** {

System.***out***.println("Le quadrilatère n'est pas un parallélogramme.");

}

System.***out***.println("Nombre de polygones construits: " + Polygone.*comptpoly*);

}

}

-------------------------------------------------------------------------------------

**public** **class** PolyLine {

**private** Point2D[] points;

**private** **int** Max;

**private** **int** nbdispo;

// Constructeur pour la création de ligne brisée avec une capacité maximale donnée

**public** PolyLine(**int** Max) {

**this**.Max = Max;

**this**.points = **new** Point2D[Max];

**this**.nbdispo = 0;

}

// 2 .

// Méthode pour ajouter un point à la ligne brisée

**public** **void** add(Point2D pt) {

**if** (nbdispo < Max) {

points[nbdispo] = pt;

nbdispo++;

} **else** {

System.***out***.println("Capacité maximale atteinte, impossible d'ajouter un nouveau point.");

}

}

// Méthode pour afficher les points de la ligne brisée

**public** **void** afficherPoints() {

System.***out***.println("Points de la ligne brisée:");

**for** (**int** i = 0; i < nbdispo; i++) {

System.***out***.println("Point " + (i + 1) + ": " + points[i]);

}

}

// 3.

// Surcharge de la méthode add pour ajouter un nombre quelconque de points

**public** **void** add(Point2D... nouveauxPoints) {

**for** (Point2D point : nouveauxPoints) {

**if** (nbdispo < Max) {

points[nbdispo] = point;

nbdispo++;

} **else** {

System.***out***.println("Capacité maximale atteinte, impossible d'ajouter de nouveaux points.");

**return**;

}

}

}

// 4.

// Méthode pour obtenir la capacité de la ligne brisée

**public** **int** pointCapacity() {

**return** **this**.Max;

}

// Méthode pour obtenir le nombre de points actuellement sur la ligne

**public** **int** nbPoints() {

**return** **this**.nbdispo;

}

// Méthode pour vérifier si un point est contenu dans la ligne brisée

**public** **boolean** contains(Point2D point) {

**for** (Point2D p : points) {

**if** (p != **null** && p.equals(point)) {

**return** **true**;

}

}

**return** **false**;

}

}

---------------------------------------------------------------------------

**public** **class** Quadrilatere **extends** Polygone {

**public** Quadrilatere(Point2D a, Point2D b, Point2D c, Point2D d) {

**super**(**new** Point2D[] { a, b, c, d });

}

@Override

**public** String nom() {

**return** "Quadrilatère";

}

**boolean** estParallelogramme() {

**double**[] cotes = **new** **double**[4];

**for** (**int** i = 0; i < 4; i++) {

cotes[i] = *longueur*(sommets[i], sommets[(i + 1) % 4]);

}

**return** (cotes[0] == cotes[2] && cotes[1] == cotes[3]) || (cotes[0] == cotes[1] && cotes[2] == cotes[3]);

}

@Override

**public** **boolean** equals(Object that) {

**if** (!**super**.equals(that))

**return** **false**;

**return** **true**;

}

}

------------------------------------------------------------------------------

**public** **class** Tableau {

// 1 //

**public** **static** **void** initialise(**int** T[])

{

**for**(**int** i=0;i<T.length;i++)

{

T[i]=-1;

}

}

// 2 //

**public** **static** **void** afficheTableau(**int** T[])

{

**for**( **int** i:T)

{

System.***out***.print(i + " ");

}

}

// 3 //

**public** **static** **int** maxTableau(**int**[] T) {

**if** (T.length == 0) {

**return** -1;}

**int** max = T[0];

**for** (**int** valeur : T) {

**if** (valeur == -1 ) **break**;

**if** (valeur > max) {

max = valeur;}

}

**return** max;

}

// 4//

**public** **static** **boolean** ajouterElement(**int** []t, **int** z)

{

// le tableau est plein //

**if** (t[t.length-1]!=-1) {

System.***out***.println("Votre tableau est deja plein");

**return**(**false**);}

// le tableau a un seul valeur //

**if** (t[0]==-1)

{

t[0]=z;

**return** (**true**);

}

**int** j=1;

// le tableau est vide //

**while** (t[j]!=-1)

{

j++;

}

t[j]=z;

**return** (**true**);

}

// 5 //

**public** **static** **int** rechercher(**int**[] tableau, **int** element) {

**for** (**int** i = 0; i < tableau.length; i++) {

**if** (tableau[i] == element) {

**return** i;}

}**return** -1;

}

// 6 //

**public** **static** **int**[] fusion(**int**[] t1, **int**[] t2) {

**int**[] resultat = **new** **int**[t1.length + t2.length];

**int** i = 0;

**for** (**int** valeur : t1) {

resultat[i] = valeur;

i++;

}

**for** (**int** valeur : t2) {

resultat[i] = valeur;

i++;

}

**return** resultat;

}

}

----------------------------------------------------------------------------

**public** **class** TableauBis {

**int**[] T;

**public** TableauBis(**int**[] tableau) {

**this**.T = tableau;

}

**public** **void** initialise() {

Tableau.*initialise*(**this**.T);

}

**public** **void** afficheTableau() {

Tableau.*afficheTableau*(**this**.T);

}

**public** **int** maxTableau() {

**return** Tableau.*maxTableau*(**this**.T);

}

**public** **boolean** ajouterElement(**int** element) {

**return** Tableau.*ajouterElement*(**this**.T, element);

}

**public** **int** rechercher(**int** element) {

**return** Tableau.*rechercher*(**this**.T, element);

}

**public** **void** fusion(**int**[] t1) {

**int**[] tableauFusion = Tableau.*fusion*(**this**.T, t1);

**this**.T = tableauFusion;

}

**public** **static** **void** main(String[] args) {

**int**[] tableauOriginal = {13, 50, 7, 1, -1, -1};

TableauBis tableauBis = **new** TableauBis(tableauOriginal);

tableauBis.afficheTableau();

**int**[] tableauAajouter = {81, 7, 5, -1, -1};

tableauBis.fusion(tableauAajouter);

tableauBis.afficheTableau();

}

}

----------------------------------------------------------------------------------

**public** **class** test {

**public** **static** **void** main(String[] args) {

Affichable[] personnes = **new** Affichable[4];

personnes[0] = **new** Developpeur("Alice", **null**, **null**, **null**, 4000);

personnes[1] = **new** Developpeur("Bob", 4500);

personnes[2] = **new** Manager("Charlie", 6000, "Informatique");

personnes[3] = **new** Manager("David", 6500, "Finance");

**for** (Affichable personne : personnes) {

personne.afficher();

}

}

-------------------------------------------------------------------------------------

**public** **class** TestPoint {

**public** **static** **void** main(String[] args) {

// Création de 3 points

Point2D p1 = **new** Point2D();

Point2D p2 = **new** Point2D(100, 100);

Point2D p3 = **new** Point2D(p2);

// Affichage des caractéristiques de chacun des points

System.***out***.println("Point p1: " + p1);

System.***out***.println("Point p2: " + p2);

System.***out***.println("Point p3: " + p3);

// Modifier le positionnement de p3 à 10,14

p3.setX(10);

p3.setY(14);

// Translater p2 de 4,4

p2.translate(4, 4);

// Modifier l'abscisse de p1 à 5

p1.setX(5);

// Affichage des 3 points après les modifications

System.***out***.println("Points après les modifications:");

System.***out***.println("Point p1: " + p1);

System.***out***.println("Point p2: " + p2);

System.***out***.println("Point p3: " + p3);

// Test de l'égalité des points p2 et p3

System.***out***.println("\"p2==p3\"=" + (p2 == p3));

// Test de l'égalité des points p3 et p1

System.***out***.println("\"p3==p1\"=" + (p3 == p1));

}

}

------------------------------------------------------------------------------------

**public** **class** TestTableau{

**public** **static** **void** main(String[] args) {

**int**[] tableau = **new** **int**[5];

Tableau.*initialise*(tableau);

System.***out***.println("Mon Tableau :");

Tableau.*afficheTableau*(tableau);

System.***out***.println(" ");

Tableau.*maxTableau*(tableau);

Tableau.*ajouterElement*(tableau,3);

Tableau.*afficheTableau*(tableau);

System.***out***.println(" ");

Tableau.*rechercher*(tableau, 3);

//1ere tableau

System.***out***.println("t1");

**int** t1[]=**new** **int** [6];

Tableau.*initialise*(t1);

Tableau.*ajouterElement*(t1,15);

Tableau.*ajouterElement*(t1,3);

Tableau.*ajouterElement*(t1,6);

Tableau.*afficheTableau*(t1);

System.***out***.println();

//2eme tableau

System.***out***.println("t2");

**int** t2 [] =**new** **int** [4];

Tableau.*initialise*(t2);

Tableau.*ajouterElement*(t2,81);

Tableau.*ajouterElement*(t2,4);

Tableau.*ajouterElement*(t1,5);

Tableau.*afficheTableau*(t2);

}

}

-----------------------------------------------------------------------------------

**import** java.util.Arrays;

**public** **class** Triangle **extends** Polygone {

**public** Triangle(Point2D a, Point2D b, Point2D c) {

**super**(**new** Point2D[] { a, b, c });

}

@Override

**public** String nom() {

**return** "Triangle";

}

**boolean** estTriangleRectangle() {

**double**[] cotes = **new** **double**[3];

cotes[0] = Polygone.*longueur*(sommets[0], sommets[1]);

cotes[1] = Polygone.*longueur*(sommets[1], sommets[2]);

cotes[2] = Polygone.*longueur*(sommets[2], sommets[0]);

Arrays.*sort*(cotes);

**return** Math.*pow*(cotes[2], 2) == (Math.*pow*(cotes[0], 2) + Math.*pow*(cotes[1], 2)) ; // Utilisation d'une marge d'erreur

}

@Override

**public** **boolean** equals(Object that) {

**if** (!**super**.equals(that))

**return** **false**;

// Additional checks specific to Triangle, if any

**return** **true**;

}

}