

DOSSIER DE CANDIDATURE

Soumis à

La commission Nationale de
Recrutement des Maîtres assistants

Dans la discipline

Informatique

Cours Programmation Orienté Objet
avec un Examen Corrigé

Elaboré par :

Dr. Ghassen HAMDI
Laboratoire MARS, université de Sousse



COURS

Programmation Orienté Objet

Niveau : 3^{ème} Génie Mécanique et Productique

Préparé Par : HAMDI Ghassen

Docteur en Sciences Informatique.

Année universitaire: 2021/2022

Descriptif du module
« Programmation orienté objet »

Objectif(s):

- ☐ Appréhender la signification de la programmation orientée objet et ses divers concepts tels que l'encapsulation, la surcharge, le polymorphisme et l'héritage.
- ☐ Acquérir la pratique de l'utilisation de structures conditionnelles et de boucles itératives.
- ☐ Rédiger des algorithmes en suivant une syntaxe clairement définie.
- ☐ Travailler avec des classes, des objets et maîtriser les concepts associés à ces deux notions.

Programme:

- Chapitre 1- Introduction à la POO et à JAVA
- Chapitre 2- Notions de base de la programmation JAVA
- Chapitre 3- La POO avec JAVA
- Chapitre 4- Composition, Association et Héritage en Java
- Chapitre 5- Classes abstraites et Interfaces en Java

Bibliographie:

- www.cours-gratuit.com.
- firmforme.be.
- jmdoudoux.developpez.com.
- www.commentcamarche.net
- esmanick.unblog.fr

Sommaire

Sommaire

Chapitre 1 : Introduction à la programmation orienté objet et à JAVA	5
I. Définition	5
II. Un bref aperçu historique ...	5
III. POO – Objet	5
IV. POO – Classe	6
V. POO – Paquetage	6
VI. POO – Modélisation	6
VII. Concepts de la POO – Héritage	7
VIII. Concepts de la POO – Polymorphisme	7
IX. Concepts de la POO - Encapsulation	8
X. L'évolution de Java au fil du temps	8
XI. Caractéristiques de Java	8
XII. Instruments requis	9
XIII. JDK ?	9
Chapitre 2 : Eléments de base de la programmation JAVA	10
I. Fondements essentiels	10
II. Commentaires	10
III. Identificateurs	10
IV. Variables	10
V. Types primitifs	11
VI. Enveloppeurs	11
VII. Opérateurs	12
VIII. Structures conditionnelles	13
1. <i>Les principes du langage C demeurent toujours applicables</i>	13
2. <i>switch...case</i>	13
3. <i>L'opérateur conditionnel ?</i>	14
IX. Structures itératives	14
1. <i>for</i>	14
2. <i>foreach</i>	14
3. <i>while</i>	15
4. <i>do ... while</i>	15
X. Saisie des données	15

1.	<i>Paramètre args de main()</i>	15
2.	<i>Les objets « Scanner »</i>	16
XI.	Chaînes de caractères	16
1.	<i>Text Block</i>	17
2.	<i>Quelques opérations</i>	18
XII.	Tableaux	18
1.	<i>Déclaration</i>	18
2.	<i>Initialisation explicite</i>	19
3.	<i>Initialisation automatique</i>	19
4.	<i>Parcours</i>	19
XIII.	Collections	19
I.	Etude du premier programme Java	21
II.	Définition des classes	22
5.	<i>Les modificateurs</i>	22
III.	Déclaration des attributs	22
IV.	Définition des méthodes	22
V.	Surcharge des méthodes	23
VI.	Création d'objets	23
VII.	Manipulation des membres d'un objet	24
VIII.	Constructeurs	24
1.	<i>Définition</i>	24
2.	<i>Surcharge</i>	25
IX.	Accesseurs	26
X.	Mutateurs	26
XI.	L'objet « null »	27
XII.	Le terme « static »	28
XIII.	Le terme « final »	28
1.	<i>Les variables « final »</i>	29
2.	<i>Les méthodes « final »</i>	29
3.	<i>Les classes « final »</i>	29
	Chapitre 4 : Héritage en Java	30
I.	Principe	30
II.	Mise en œuvre	30
	Chapitre 5 : Classes abstraites et Interfaces en Java	35
I.	Usage de l'abstraction	35
1.	<i>Surface des formes géométriques</i>	35
a.	Mise en œuvre de méthode abstraite	36
II.	Directives pour l'emploi des classes abstraites	37
III.	Concept d'Interface	37

IV. Traits des interfaces	37
2. <i>Syntaxe de définition</i>	37
3. <i>Syntaxe d'implémentation</i>	38
V. Héritage multiple	38

Chapitre 1 : Introduction à la programmation orienté objet et à JAVA

I. Définition

La programmation orientée objets (POO) est définie pour la modélisation d'un regroupement d'éléments provenant d'une portion du monde concret en un groupe d'éléments informatiques que l'on appelle objets.

- Il y a des données informatiques qui regroupent les traits dominants des éléments présents dans la réalité tangible (dimensions, teintes, etc.).
- Chaque élément exprime une occurrence spécifique d'une classe donnée. Ces classes ont la capacité d'appartenir à un ensemble ou une structure hiérarchique de classes interconnectées entre elles au moyen de liens.

II. Un bref aperçu historique ...

- Simula (1967) marque l'avènement du premier langage de programmation qui a implémenté l'émergence du concept de classes en 1967.
- Smalltalk (1972-1980) a intégré les concepts fondamentaux de l'approche objet, tels que l'encapsulation, l'agrégation et l'héritage.
- Les années 80 ont été marquées par l'essor des langages de programmation qui adoptent une approche orientée objet.

C++ (1983)	Objective-C (1984)
Eiffel (1986)	Common Lisp Object System (1988)

- La décennie des années 90 marque l'apogée de la programmation orientée objet avec l'avènement de Java en 1995, couvrant des alternatives tant sur les ordinateurs de bureau que sur le Web.
- En l'an 2000, émergence du langage phare de Microsoft : C#.

III. POO – Objet

NB : Les expressions "instance" et "objet" sont utilisés de manière interchangeable.

- L'objet peut être défini par plusieurs concepts, incluant un état, un comportement et une identité ;
 - ❖ L'état : se caractérise par un ensemble d'attributs destinés à stocker des données concernant l'objet, tels que sa taille, sa couleur, sa marque, son âge, etc.
 - ❖ L'identité : sert à distinguer un objet des autres.
 - À titre d'exemples, un numéro de série pourrait être attribué à une voiture, tandis qu'une personne pourrait être identifiée par un numéro de carte d'identité nationale, et ainsi de suite.
 - ❖ Le comportement : les méthodes définissent le comportement de l'objet.

- Ces méthodes représentent les diverses actions ou opérations que l'objet est capable d'effectuer.
- Ces opérations sont intimement associées aux attributs, car elles peuvent être influencées par les valeurs de ces derniers et également les modifier.
 - **Démarrer** et **Freiner** dans le cas d'une voiture.
 - **Émettre des miaulements** pour un chat.
 - **Effectuer un retrait d'argent** pour un compte bancaire.
 - **Organiser** les éléments d'un tableau.

IV. POO – Classe

- Une classe représente une conceptualisation regroupant des objets partageant une organisation et une conduite communs.
 - ❖ Une voiture comme exemple de classe
 - Les divers objets de la catégorie "Voiture" se différencient au minimum par l'un de leurs attributs.
- La classe est constituée de deux éléments :
 - ❖ Les attributs : les données qui reflètent l'état de l'objet.
 - ❖ Les méthodes : les actions pouvant être effectuées sur les objets.

V. POO – Paquetage

- Il est impossible de rencontrer deux classes portant le même nom dans un même fichier ou dossier.
- Il est possible d'avoir deux classes distinctes portant le même nom dans deux répertoires différents.
 - ❖ Comment superviser celle que l'on souhaite utiliser ?
 - En utilisant le concept de package.
- Le package constitue un répertoire rassemblant des classes liées par un domaine similaire.
- Chaque classe qui se trouve au sein des fichiers présents dans ce répertoire constituera un élément essentiel du package.
- Les packages sont utilisés pour résoudre les conflits de noms.

VI. POO – Modélisation

- Implique la création d'une représentation graphique des composants de la réalité sans leur mise en œuvre
 - ❖ Cela implique une approche indépendante des langages de programmation et des technologies utilisées.
- De 1970 à 1990, plusieurs méthodologies orientées objet ont été créées, à tel point qu'en 1994, on en comptait plus de 50.
- Seules trois méthodes ont réellement pris de l'ampleur
 - ❖ La méthode OMT de Rumbaugh
 - ❖ La méthode BOOCH'93 de Booch
 - ❖ La méthode OOSE de Jacobson

- En 1995, ce groupe a développé le langage UML (Unified Modeling Language)
 - ❖ Langage standard, incluant les bénéfices des diverses méthodes évoquées précédemment.

VII. Concepts de la POO – Héritage

- Facilite la spécialisation en permettant la création de nouvelles classes dérivées en se basant sur une classe déjà existante.
 - ❖ La classe fraîchement instaurée, qualifiée de classe dérivée, englobe les attributs et les méthodes hérités de sa classe principale.
 - ❖ La sous-classe introduit des attributs et méthodes supplémentaires.
 - Cela facilite l'établissement d'une structure de classes en hiérarchie de plus en plus dédiées sans avoir à recommencer depuis le début.
- Héritage multiple : Dans Des langages qui adoptent une approche orientée objet tels que le C++ et le C#, il est possible d'implémenter la possibilité d'hériter de plusieurs superclasses grâce à la notion d'héritage de plusieurs sources.
 - ❖ En Java : ce qu'on appelle "l'héritage de plusieurs sources" n'est pas présente.

Exemples

- ❖ Un avocat, un vendeur et un enseignant appartiennent à la catégorie des personnes.
- ❖ Un enseignant permanent et un enseignant vacataire sont tous deux considérés comme des individus.
- ❖ Un enseignant vacataire possède cinq caractéristiques.
- ❖ Les animaux carnivores, herbivores et omnivores font partie de la classification des animaux.
- ❖ Les omnivores présentent à la fois des caractéristiques de carnivores et d'herbivores.

VIII. Concepts de la POO – Polymorphisme

- 3 types
 - ❖ (1) Polymorphisme ad hoc (surcharge) : la présence de méthodes partageant un nom commun et offrant des fonctionnalités similaires au sein de classes non liées entre elles.
 - Il est possible qu'une méthode nommée "afficher" soit présente au sein de diverses classes.
 - ❖ (2) Dans le contexte du polymorphisme d'héritage (redéfinition ou spécialisation), on ajuste une méthode en la réécrivant au sein des classes qui héritent de la classe originale.
 - ❖ (3) Le polymorphisme paramétrique (généricité) se manifeste lorsque plusieurs méthodes, ayant le même nom, sont définies avec des paramètres distincts en termes de nombre et/ou de type.
 - somme() réalisant une addition de valeurs.
 - La méthode *int somme(int, int)* retourne l'addition de deux entiers
 - La méthode *int somme(int, int, int)* retourne l'addition de trois entiers

- La méthode *float somme(float, float)* retourne l'addition de deux flottants
 - La signature, qui inclut à la fois le nombre et le type des arguments, est déterminante pour identifier quelle méthode sera appelée.

IX. Concepts de la POO - Encapsulation

- Un dispositif visant à restreindre la manipulation des données en dehors des méthodes fournies.
 - ❖ Assure que les données stockées dans l'objet sont cohérentes.
- Offre la possibilité d'établir des degrés de visibilité qui déterminent les permissions du droit d'entrée aux composants de la classe en fonction de l'accès depuis la classe elle-même, une classe dérivée, une classe du paquetage d'origine ou toute autre classe.

	Même classe	Même Paquetage	Sous classe	Extérieur
PUBLIC	<i>Oui</i>	<i>Oui</i>	<i>Oui</i>	<i>Oui</i>
PROTECTED	<i>Oui</i>	<i>Oui</i>	<i>Oui</i>	<i>Non</i>
PRIVATE	<i>Oui</i>	<i>Non</i>	<i>Non</i>	<i>Non</i>
Rien	<i>Oui</i>	<i>Oui</i>	<i>Non</i>	<i>Non</i>

X. L'évolution de Java au fil du temps

- En 1991, Sun Microsystems a conçu Java.
 - ❖ Objectif : Incorporer Java dans les dispositifs électroménagers dans le but de les superviser, les doter de fonctionnalités interactives et favoriser la communication entre eux.
- À l'origine, le langage était appelé Oak, signifiant chêne.
 - ❖ Une part significative de la production de café a son origine sur l'île Java, qui est nommée d'après le breuvage favori des développeurs, le café.
- Officiellement dévoilé lors de la conférence SunWorld en 1995.
- En 2009, Oracle a acquis Sun.
 - ❖ Elle possède et assure désormais la maintenance de Java.
- Actuellement, nous en sommes déjà à la version 15.

XI. Caractéristiques de Java

- Java se distingue par son orientation objet, permettant de traiter non seulement des nombres, des caractères, des tableaux, etc., mais aussi des objets du monde réel.
- Java est facile à apprendre : sa syntaxe est proche de celle de C et de C++, et il gère efficacement les pointeurs.
- Java est un langage interprétable, cela implique qu'il doit être transformé en code

machine et traduit en bytecode avant d'être exécuté. Le processeur ne peut pas exécuter directement le bytecode, il doit être traduit en binaire par un interpréteur (JVM).

- Java est multi-plateforme : grâce à la présence de JVM, une application développée en Java peut s'exécuter dans divers environnements.
- Java est extensible : toutes les classes élaborées pour résoudre une problématique spécifique ont la possibilité d'être intégrées au langage et employées pour adresser de nouvelles problématiques.

XII. Instruments requis

- JDK (Java Development Kit)
 - ❖ <https://www.oracle.com/java/technologies/javase-downloads.html>
- Un environnement de développement intégré parmi ceux énumérés ci-dessous (liste non exhaustive)
 - ❖ Eclipse IDE 2020
 - <https://www.eclipse.org/downloads/>
 - ❖ NetBeans
 - <https://netbeans.apache.org/download/index.html>
 - ❖ IntelliJ IDEA
 - <https://www.jetbrains.com/idea/download/>

XIII. JDK ?

Cet ensemble d'outils de développement englobe une variété d'éléments, parmi lesquels :

- javac : l'outil de compilation Java
- java : l'exécuteur des bytecodes Java, également désigné sous le terme JVM.
- jdb : l'outil de débogage Java, plus accessible lorsqu'il est utilisé à partir d'un environnement de développement intégré (EDI) tel que Eclipse, NetBeans, etc.
- javap : l'outil de décompilation Java, permettant de retrouver le code source à partir du bytecode
- javadoc : l'outil de génération de documentation au style HTML, utilisant les commentaires et les balises spéciales telles que @author, @param, @return, etc.
- jar : l'outil de compression de fichiers Java.
- jarsigner : Un outil indispensable qui assure la signature et la vérification des fichiers d'archive, jouant un rôle crucial en termes de sécurité et d'efficacité..

Chapitre 2 : Eléments de base de la programmation JAVA

I. Fondements essentiels

- À l'instar du langage C
 - ❖ Java fait la différence entre la majuscule et la minuscule.
 - ❖ Des accolades encadrent les segments de code.
 - ❖ Il faut que chaque instruction se conclut avec le caractère ';' (point-virgule).
 - ❖ Il est possible que chaque instruction s'étende sur plusieurs lignes.

```
char
code
=
'D' ;
```

II. Commentaires

Type de commentaires	Exemple
commentaire abrégé	// commentaire sur une seule ligne int N=1; // déclaration du compteur
commentaire multi-ligne	/* commentaires ligne 1 commentaires ligne 2 */
commentaire de documentation automatique (utilisé par l'utilitaire <i>javadoc</i>)	/** * commentaire de la méthode * @param val la valeur à traiter * @since 1.0 * @return Rien * @deprecated Utiliser la nouvelle méthode XXX */

III. Identificateurs

- Chacun des aspects suivants : un nom, appelé « identificateur », est associé à chaque objet, classe, programme ou variable.
 - ❖ peut-être l'ensemble des caractères alphanumériques, y compris les caractères « _ » et « \$ ».
- La première lettre nécessite un caractère alphabétique.
- Il faut que l'identificateur n'appartienne pas à l'ensemble des mots clés du langage Java.

IV. Variables

- Une variable se définit par un nom, un type, et une valeur associée. Lors de sa spécification, il est obligatoire de préciser simultanément son nom et son format de données. Une variable est opérationnelle uniquement dans la portion de code où elle a été déclarée.
- L'acte de déclarer une variable permet de :

- ❖ Allouer de l'espace mémoire pour conserver la valeur.
- ❖ Définir les valeurs envisageables.
- ❖ Établir les opérateurs / opérations envisageables.
- La nature d'une variable peut se présenter sous la forme de :
 - ❖ un type de base ou primitif :
 - float e ;
 - ❖ une classe
 - Voiture v ;
- Attribution des valeurs initiales aux variables

```
int nombre1; // déclaration
nombre1 = 100; //initialisation

int nombre2 = 500; //déclaration et initialisation
```

V. Types primitifs

Primitive	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	2	de 0 à 65535
byte	Entier très court	1	de -128 à 127
short	Entier court	2	de -32768 à 32767
int	Entier	4	de -2 147 483 648 à 2 147 483 647
long	Entier long	8	de -9223372036854775808 à 9223372036854775807
float	flottant (réel)	4	de $-1.4 \cdot 10^{-45}$ à $3.4 \cdot 10^{38}$
double	flottant double	8	de $4.9 \cdot 10^{-324}$ à $1.7 \cdot 10^{308}$
boolean	Booléen	1	0 (<i>false</i>) ou 1 (<i>true</i>) « en réalité, toute autre valeur que 0 est considérée égale à 1 »
void	Rien	0	-

VI. Enveloppeurs

- Il est possible d'encapsuler les types primitifs dans des objets créés pour cette raison, appelés 'enveloppeurs' (wrappers).
- Les programmeurs sont des éléments qui peuvent englober une donnée de base, où ils attachent des méthodes pour la manœuvrer.

Enveloppeur	Primitive associée
BigDecimal	-
BigInteger	-
Character	char
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Boolean	boolean
Void	void

Exemple

```
Integer int_wrapper =10;
int_wrapper.
```

- byteValue() : byte - Integer
- compareTo(Integer anotherInteger) : int - Integer
- describeConstable() : Optional<Integer> - Integer
- doubleValue() : double - Integer
- equals(Object obj) : boolean - Integer
- floatValue() : float - Integer
- getClass() : Class<?> - Object
- hashCode() : int - Integer
- intValue() : int - Integer
- longValue() : long - Integer
- notify() : void - Object
- notifyAll() : void - Object

Press 'Ctrl+Space' to show Template Proposals

```
Integer int_wrapper =10;
System.out.println("MAX_VALUE = "+int_wrapper.MAX_VALUE);
System.out.println("MIN_VALUE = "+int_wrapper.MIN_VALUE);
```

```
MAX_VALUE = 2147483647
MIN_VALUE = -2147483648
```

VII. Opérateurs

- Arithmétiques : +, -, *, /, %
- Affectation : =, +=, -=, *=, /= et %=
- Incrémentation : ++
- Décrémententation : --
- Comparaison : >, >=, <, <=, ==, !=
- Opérateurs logiques : &&, ||

VIII. Structures conditionnelles

1. Les principes du langage C demeurent toujours applicables

- if
- if ... else
- if imbriqués
- switch

2. switch...case

Syntaxe :

```
switch (i) {  
  case 0 :  
    System.out.println("Traitement du 0"); break;  
  case 1:  
    System.out.println("Traitement du 1"); break;  
  case 2:  
    System.out.println("Traitement du 2"); break;  
  default: System.out.println("Aucun traitement associé");  
}
```

La nouvelle directive switch (introduite avec Java SE 14)

```
switch (i) {  
  case 0 ->  
    System.out.println("Traitement du 0");  
  case 1->  
    System.out.println("Traitement du 1");  
  case 2->  
    System.out.println("Traitement du 2");  
  default->System.out.println("Aucun traitement associé");  
}
```

Le mot clé « *yield* » : pertinent lorsqu'on emploie le switch dans une expression.


```
String m=switch (i) {
    case 0 ->
        {System.out.println("Traitement du 0");
        yield "ZERO";
        }
    case 1->
        {System.out.println("Traitement du 1");
        yield "UN";
        }
    case 2->{
        System.out.println("Traitement du 2");
        yield "DEUX";
        }
    default-> {
        System.out.println("Aucun traitement associé");
        yield "RIEN";
        }
};

System.out.println(m);
```

3. L'opérateur conditionnel ?

Cela constitue opérateur ternaire

```
int entier = 10;
String msg = entier==10 ? "OK" : "Not OK";
System.out.println(msg);
```

IX. Structures itératives

Les règles du langage C restent également valides.

1. for

Syntaxe

```
for (int i = 1; i <= 10; i++) {
}
```

2. foreach

Syntaxe

```
for( int value : intArray ) {
    System.out.println( value );
}
```

Remarque

- Le terme foreach n'est pas pris en charge dans java. Cette boucle est implémentée

à l'aide du terme `for`, tout en ayant une écriture légèrement différente.

- Son utilisation principale est la traversée de tableaux ou de collections.

3. *while*

Syntaxe

```
while (entier < 100) {  
    entier++;  
}
```

4. *do ... while*

Syntaxe

```
do {  
    entier++;  
}while(entier<100);
```

X. Saisie des données

1. Paramètre *args* de *main()*

Ce paramètre symbolise un tableau d'arguments destinés à être passés en ligne de commande.

```
public class MainArgs {  
  
    public static void main(String[] args) {  
        for(String s : args){  
            System.out.println(s);  
        }  
    }  
}
```

```
D:\eclipse-workspace\HelloIAM\src>javac MainArgs.java  
  
D:\eclipse-workspace\HelloIAM\src>java MainArgs  
  
D:\eclipse-workspace\HelloIAM\src>java MainArgs arg1 25 true  
arg1  
25  
true
```

Exemple d'arguments entiers à traiter

Pour effectuer leur somme, une conversion en type « `int` » est indispensable.

```
public class MainArgs {
    public static void main(String[] args) {
        // les deux arguments sont des entiers
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);
        System.out.print("La somme des arguments est " + (x+y));
    }
}
```

```
D:\eclipse-workspace\HelloIAM\src>javac MainArgs.java
D:\eclipse-workspace\HelloIAM\src>java MainArgs 15 6
La somme des arguments est 21
D:\eclipse-workspace\HelloIAM\src>java MainArgs 15 a
Exception in thread "main" java.lang.NumberFormatException: For input string: "a"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:68)
    at java.base/java.lang.Integer.parseInt(Integer.java:652)
    at java.base/java.lang.Integer.parseInt(Integer.java:770)
    at MainArgs.main(MainArgs.java:7)
```

2. Les objets « Scanner »

Un objet Scanner est utilisé pour lire les données saisies à partir du clavier.

```
Scanner scanner = new Scanner( System.in );
System.out.print("Entrer votre nom : ");
String nom = scanner.nextLine();
System.out.print("Entrer votre prénom : ");
String prenom = scanner.nextLine();
System.out.print("Bonjour M. " + prenom + " " + nom);
scanner.close();
```

```
Entrer votre nom : Hamdi
Entrer votre prénom : Ghassen
Bonjour M. Hamdi Ghassen
```

```
Scanner scanner = new Scanner( System.in );
System.out.print("Entrer un 1er entier : ");
int i1 = scanner.nextInt();
System.out.print("Entrer un 2ème entier : ");
int i2 = scanner.nextInt();
System.out.print("La somme de "+i1+" et "+i2+" est : "+(i1+i2));
scanner.close();
```

```
Entrer un 1er entier : 15
Entrer un 2ème entier : -8
La somme de 15 et -8 est : 7
```

XI. Chaînes de caractères

- Contrairement aux types primitifs, les chaînes de caractères appartiennent à la classe String.
- On peut définir une chaîne de caractères comme étant un objet qui dispose de méthodes et d'attributs.

Déclaration

```
String s = "Bonjour";
```

1. Text Block

Depuis la version Java SE 15, un nouveau type de chaîne a été ajouté : les chaînes de caractères multilignes, également connues sous le nom de "Text Blocks".

Une chaîne de ce format débute avec trois guillemets et se clôture avec trois guillemets.

Exemple :

```
String multiline_text = "il s'agit d'une\n"+  
    "chaîne de caractères\n"+  
    "sur plusieurs lignes\n";
```

```
String text_block = ""  
    il s'agit d'une  
    chaîne de caractères  
    sur plusieurs lignes  
    "";
```

Pratique pour l'utilisation dans des codes Java impliquant des requêtes SQL ou des blocs HTML.

```
public String getBlockOfHtml() {  
    return ""  
        <html>  
  
        <body>  
            <span>example text</span>  
        </body>  
    </html>"";  
}
```

2. Quelques opérations

- Opérateur de concaténation : `+`
- Déterminer la taille : `s.length()`
- Comparer deux chaînes :
 - ❖ `if(s.equals(ch)) ...` // retourne true ou false.
 - ❖ `If(s.compareTo(ch)==0)` // retourne 0 si identiques, autre valeur sinon.
 - ❖ On n'utilise pas l'opérateur « `==` » car il compare des adresses (pointeurs).
- Déterminer la position d'un caractère :
 - ❖ `s.indexOf('A')` // retourne l'indice du caractère A dans la chaîne s (-1 s'il n'existe pas)
- Déterminer le caractère à l'indice donné : `s.charAt(3)` // retourne le 4ème caractère de s, une exception si l'indice est \geq à la taille de s.
- Vérification de la présence d'une sous-chaîne dans une chaîne donnée :
 - ❖ `if(s.contains("abc")) ...`
- Déterminer si une chaîne se conclut par une sous-chaîne spécifiée :
 - ❖ `if(s.endsWith(ch)) ...`
- Remplacement d'une portion de la chaîne par une autre chaîne :
 - ❖ `s.replace("abc", "xy")`
- Conversion d'une chaîne en Maj ou en min :
 - ❖ `s.toUpperCase()`
 - ❖ `s.toLowerCase()`

XII. Tableaux

- Un tableau est une structure conçue pour contenir un nombre déterminé d'éléments homogènes.
- Il est envisageable que les composants d'un tableau soient à la fois des objets ou des primitives.
 - ❖ Même lorsque les primitives sont présentes, les tableaux sont toujours considérés comme des objets.
- Pour accéder à chaque élément, il suffit de connaître son indice commençant par 0.

1. Déclaration

- Deux syntaxes équivalentes :
 - ❖ `int[] x;`
- ou :
 - ❖ `int x[];`
- Néanmoins, il convient d'être prudent lors de l'utilisation de l'une ou de l'autre des écritures, parce que :
 - ❖ `int [] x, y` ; Cela veut dire que x et y sont tous les deux, des tableaux d'entiers.
 - ❖ `int x[], y` ; On peut affirmer que seul x est un tableau d'entiers et y n'est qu'un entier.
- L'écriture équivalente à la première est :
 - ❖ `int x[], y[];`

2. Initialisation explicite

- Commencez par l'initialisation du tableau dès sa création.
- Après la spécification pas la taille du tableau.
 - ❖ `int tableau[] = {10,20,30,40,50};`
 - ❖ Java détermine automatiquement la taille (ici 5).

3. Initialisation automatique

- Mise en place d'un tableau contenant des valeurs préétablies préalablement déterminées par le système.
 - ❖ `float[] A = new float[7];`
- En utilisant cette notation, le tableau A est initialisé avec une valeur préétablie de 7 zéros.
 - ❖ Les valeurs par défaut sont :
 - ❖ floats: 0,0, booléens: false , null pour les objets, la caractère ' ' (espace) pour les char etc.
- En recourant à l'opérateur new, il est envisageable d'allouer l'espace nécessaire pour le stockage de ce tableau (7*8 octets).

4. Parcours

- L'utilisation de boucles est essentielle pour parcourir le parcours d'un tableau. Dans cet exemple, **length** renvoie le nombre de composants présents dans le tableau.

Exemple

```
int [] tab = new int[5];

tab[0]=1;
tab[1]=11;
tab[2]=21;
tab[3]=31;
tab[4]=41;

for (int i = 0; i < tab.length ; i++)
{
    System.out.println(tab[i]);
}
```

```
for (int element : tab)
{
    System.out.println(element);
}
```

XIII. Collections

- Contrairement à un tableau dont la taille doit être fixée dès sa déclaration, une collection peut être agrandie en ajoutant des éléments tant que des ressources mémoire sont disponibles.
- Les collections peuvent être implémentées de différentes manières. (ArrayList, LinkedList, Vector, Stack, PriorityQueue, etc.).
- En fonction de la classe sélectionnée, vous pourrez également choisir les algorithmes correspondants.
- `java.util.ArrayList` implémente un tableau d'éléments extensible en fonction du besoin.
- `java.util.LinkedList` implémente une liste doublement chaînée.
- `java.util.Stack` implémente une pile.
- `java.util.PriorityQueue` implémente une file.

Examples

```
ArrayList<String> students=new ArrayList<String>();
students.add("Ahmed");
students.add("Ali");
students.add("Sami");
students.add("Samia");

for(String st:students)
    System.out.println(st);
```

```
Ahmed
Ali
Sami
Samia
```

```
Stack<String> students=new Stack<String>();
students.add("Ahmed");
students.add("Ali");
students.add("Sami");
students.add("Samia");
students.pop();
students.add("Karim");
students.pop();
students.pop();

for(String st:students)
    System.out.println(st);
```

```
Ahmed
Ali
```

```
PriorityQueue<String> students=new PriorityQueue<String>();
students.add("Ahmed");
students.add("Ali");
students.add("Sami");
students.add("Samia");
String stud=students.peek();

System.out.println("stud "+ stud);

for(String st:students)
    System.out.println(st);
```

```
stud Ahmed
Ahmed
Ali
Sami
Samia
```

```
PriorityQueue<String> students=new PriorityQueue<String>();
students.add("Ahmed");
students.add("Ali");
students.add("Sami");
students.add("Samia");
String stud=students.poll();

System.out.println("stud "+ stud);

for(String st:students)
    System.out.println(st);
```

```
stud Ahmed
Ali
Samia
Sami
```

I. Etude du premier programme Java

```
/** Programme affiche "Hello World" */  
public class HelloWorld {  
    public static void main (String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

▪ Ligne 1

- ❖ définit une classe dite « First » grâce au mot clé **class**.
- ❖ Cette classe est publique donc :
 - Elle pourrait être utilisée / appelée par toute autre classe.
 - Elle doit exister dans un fichier nommé obligatoirement « First.java ».

▪ Ligne 2

- ❖ Spécifie la méthode prédominante de l'application.
- ❖ Void est un mot clé qui indique que la méthode ne retourne rien.
- ❖ Un tableau de chaînes de caractères est représenté par **String args []**.
 - Il contiendra les arguments passés au programme via une ligne de commandes.
- ❖ Le mot clé **public** implique que l'accès à la méthode **main()** est autorisé à toutes les classes.
 - Étant traité comme une classe, il est crucial que l'interpréteur Java puisse y accéder pour lancer l'application.
 - L'omission du mot-clé "public" ne causerait pas d'obstacle à la compilation normale du programme, cependant, cela entraverait son exécution.
- ❖ **static** est un mot-clé signalant que la méthode est une caractéristique intrinsèque de la classe.

▪ Ligne 3

- ❖ Utilisation de la méthode **println()**, qui affiche sur la sortie standard la chaîne de caractères fournie en paramètre, avec ajout automatique d'un saut de ligne.
- ❖ Cette méthode est précédée de **System.out.**,
- ❖ **out** est un objet de la classe **System**.

II. Définition des classes

- Dans une classe, on regroupe des attributs et des méthodes. Une classe représente une représentation abstraite d'un objet. Lors de son instantiation, on crée un objet correspondant. On peut comparer la relation entre une classe et un objet à celle entre un type et une variable.
- La déclaration d'une classe s'effectue comme suit :

```
class Nom_de_la_classe {  
    // Instructions pour la définition de la classe;  
}
```

5. Les modificateurs

- **public**
 - ❖ La classe est disponible de manière globale.
- **final**
 - ❖ La classe n'est pas sujette à l'héritage.
- **abstract**
 - ❖ La classe comporte au minimum une méthode abstraite
- **extends** est un mot clé qui sert d'indiquer un lien d'héritage.
- **implements** est un terme qui sert d'indiquer une ou des interfaces que la classe met en œuvre.
 - ❖ Cela autorise de tirer parti de certains bénéfices de l'héritage multiple.
 - ❖ L'abstraction caractérise toutes les opérations d'une interface.
- Dans une classe, La séquence des méthodes n'a aucune incidence.

III. Déclaration des attributs

- On écrit le type suivit du nom de la variable comme en C

```
public class Voiture {  
    int id;  
    public String Marque;  
    protected double vitesse;  
    private int prix;  
    public Moteur moteur;  
}
```

IV. Définition des méthodes

On définit une méthode par son contenu et son entête comme suit :

```
type_ret Nom_methode(type1 argument1, type2 argument2, ...){  
    // corps de la méthode  
}
```



```
public int somme (int x, int y)
{
    return x + y;
}
```

```
public Boolean Puis_je_acheter (double mon_argent)
{
    if(mon_argent >= prix)
        return true;
    else return false;
}
```

V. Surcharge des méthodes

C'est simplement la capacité d'invoquer plusieurs méthodes partageant le même nom, pourvu que leurs paramètres distinct de nombre et/ou en type.

```
// La méthode somme() est surchargée, elle a plusieurs signatures possibles

// Somme de deux entiers
int somme( int p1, int p2){ return p1 + p2;}

// Somme de deux réels
float somme( float p1, float p2){return p1 + p2;}

// Somme de trois réels
float somme( float p1, float p2, float p3){return p1 + p2 + p3;}

// Cette méthode est identique à la précédente, juste elle fait appel à la méthode
// Somme() de deux réels
///// float somme( float p1, float p2, float p3){return somme(p1, p2) + p3;} /////

// Somme d'un réel et d'un entier
float somme( float p1, int p2){return p1 + p2;}
```

VI. Création d'objets

- La fabrication d'objets est connue sous le nom d'instanciation.
- Pour réaliser cette instanciation, on utilise l'opérateur "new" en suivant du nom de la classe pour créer une instance, avec des parenthèses renfermant les paramètres d'instanciation.

```
Voiture v1;  
v1= new Voiture();
```

```
Voiture v2 = new Voiture();
```

```
Voiture v3 = new Voiture(100, "Volkswagen", 260, 80, new Moteur());
```

VII. Manipulation des membres d'un objet

Pour accéder aux membres de données (attributs) d'un objet, utilisez le nom de l'objet suivi d'un point, puis spécifiez le nom du membre de données, puis du nom du membre de données.

```
v.Marque="Mercedes";  
v.vitesse=240;
```

On accède aux méthodes comme suit :

```
v1.Démarrer();  
Boolean b = v3.Puis_je_acheter(90);
```

VIII. Constructeurs

- On adopte le terme **new** afin de met en œuvre des.
 - ❖ Ce terme utilise une opération spécifique nommé **constructeur**.
- Elle a pour fonction de déclarer et d'effectuer l'initialisation des attributs d'une classe.
- Un constructeur a les deux caractéristiques suivantes :
 - ❖ Il est impératif que son nom corresponde exactement à celui de la classe.
 - ❖ Il n'a pas de retour.

1. Définition

- La spécification d'un constructeur est optionnelle.
- En l'absence de spécification d'un constructeur, la machine virtuelle invoque automatiquement un constructeur sans arguments qui est automatiquement généré.
- Immédiatement un constructeur est spécifié de façon explicite, Java suppose que :
 - ❖ Le programmeur serait responsable de la conception des constructeurs
 - ❖ Désactivation du constructeur sans arguments.
- Pour préserver ce mécanisme, un constructeur sans arguments est spécifié d'une façon explicite.

Exemple

Exemple de la classe « *Rectangle* » définie par ses deux attributs « *larg* » pour *largeur* et « *lng* » pour *longueur*.

```
public class Rectangle {  
  
    double larg;  
    double lng;  
  
    public Rectangle(double larg, double lng) {  
        this.larg=larg;  
        this.lng=lng;  
    }  
    /*  
    public Rectangle(double x, double y) {  
        larg=x;  
        lng=y;  
    }  
    */  
}
```

2. Surcharge

```
public class Rectangle {  
  
    double larg;  
    double lng;  
  
    // Constructeur par défaut qui donne des valeurs nulles  
    // pour la largeur et la longueur  
    public Rectangle(){  
        larg=0;  
        lng=0;  
    }  
  
    // Constructeur qui crée un rectangle ayant une longueur  
    // égale au double de la largeur  
    public Rectangle(double dim) {  
        larg=dim;  
        lng=2*dim;  
    }  
  
    // Constructeur qui crée un rectangle avec les valeurs  
    // passées en arguments  
    public Rectangle(double larg, double lng) {  
        this.larg=larg;  
        this.lng=lng;  
    }  
}
```

Exemple d'invocations

```
public static void main(String[] args) {
    Rectangle r1= new Rectangle();
    Rectangle r2= new Rectangle(3.2);
    Rectangle r3= new Rectangle(5.1, 7.3);

    System.out.println("R1 (" +r1.larg+", " +r1.lng+"");
    System.out.println("R2 (" +r2.larg+", " +r2.lng+"");
    System.out.println("R3 (" +r3.larg+", " +r3.lng+"");
}
```

```
R1 (0.0, 0.0)
R2 (3.2, 6.4)
R3 (5.1, 7.3)
```

IX. Accesseurs

- Un accesseur :
 - ❖ Devrait retourner un type de l'attribut à récupérer.
 - ❖ Sans arguments.
- Les accesseurs débutent par get.

Exemples

```
private double larg;
private double lng;

public double getLarg() {
    return larg;
}

public double getLng() {
    return lng;
}
```

```
private String brand;
private boolean electric;

public String getBrand() {
    return brand;
}

public boolean isElectric() {
    return electric;
}
```

X. Mutateurs

- Une méthode connue sous le nom de mutateur est utilisée pour altérer la valeur d'un attribut privé.
- Un mutateur :
 - ❖ La valeur à assigner à l'attribut doit être un de ces paramètres.

- ❖ Ne retourne pas obligatoirement une valeur.
- Les mutateurs débutent par **set**. On les désigne également sous le nom de **setters**.

Exemple

```
public class CompteBancaire {  
    private String numCompte;  
    private double solde;  
  
    public String getNumCompte() {  
        return numCompte;  
    }  
  
    public double getSolde() {  
        return solde;  
    }  
  
    public void setNumCompte(String newNum) {  
        numCompte = newNum;  
    }  
  
    public void setSolde(double newSolde) {  
        if (newSolde >= 0) solde = newSolde;  
        else System.out.println("Opération non autorisée");  
    }  
}
```

```
public static void main(String[] args) {  
    CompteBancaire cb = new CompteBancaire();  
    cb.setSolde(0);  
    System.out.println("Solde initial = " + cb.getSolde());  
    cb.setSolde(100);  
    System.out.println("Nouveau solde = " + cb.getSolde());  
    cb.setSolde(-10);  
    System.out.println("Nouveau solde = " + cb.getSolde());  
}
```

```
Solde initial = 0.0  
Nouveau solde = 100.0  
Opération non autorisée  
Nouveau solde = 100.0
```

XI. L'objet « null »

- On peut l'utiliser en remplacement d'un objet de toute classe ou comme paramètre, même s'il n'appartient pas à une classe.
- Pour libérer la mémoire allouée à un objet, il suffit d'initialiser une variable référençant cet objet à null.

- En cas de besoin d'un objet qui est null, une exception `NullPointerException` est générée par le programme.

XII. Le terme « static »

- Il mentionne les variables de la classe et les méthodes de la classe.
 - ❖ Il convient de souligner que ces variables et méthodes ne sont pas propres à chaque objet.

```
public class Chat {
    public String nom;
    public String race;
    public static int nbPattes;
}
```

« nom » et « race » sont des variables d'instance : on peut avoir des valeurs différentes pour chaque objet.

- « nbPattes » est une variable qui ne se change pas par les objets..
 - ❖ La mémoire contient une seule variable de classe, indépendamment du nombre d'objets créés. Si elle est modifiée, cette variable est répercutée sur toutes les instances.
- Une méthode statique ne peut modifier que des variables de classe (ou des attributs statiques).
- On utilise cette écriture `classe.methode()` et n'est pas `objet.methode()`.

Exemple

```
public static void doSomething()
{
    // Traitement quelconque
}

public static void main(String args[])
{
    Chat c1=new Chat();
    Chat.nbPattes=7;
    Chat.doSomething();

    c1.doSomething();
}
```

The static method `doSomething()` from the type `Chat` should be accessed in a static way

4 quick fixes available:

- [Change access to static using 'Chat' \(declaring type\)](#)
- [Remove 'static' modifier of 'doSomething\(\)'](#)
- @ [Add @SuppressWarnings 'static-access' to 'main\(\)'](#)
- 🔧 [Configure problem severity](#)

Press 'F2' for focus

XIII. Le terme « final »

- Il garantit que l'entité à laquelle il est appliqué ne peut pas être modifiée après sa déclaration (par exemple, une méthode ou une classe) ou son initialisation (par

exemple, une variable).

1. Les variables « final »

- Une variable qui a été déclarée comme final signifie qu'une fois initialisée, elle devient immuable et ne peut plus subir de modifications.
 - ❖ Cela indique que nous sommes en présence d'une constante.
Public final int cnst1 = 20 ;
 - ❖ Les constantes sont souvent caractérisées par l'utilisation des modificateurs "final" et "static".
Public static final float pi= 3.141;

2. Les méthodes « final »

- Aucune redéfinition ne peut leur être imposée au sein d'une classe fils.

3. Les classes « final »

- Il n'est pas autorisé de générer une classe fils qui bénéficie de l'héritage de cela.
- Le terme « **this** » : il s'agit d'une variable système utilisée pour faire référence à l'objet actuel.

Chapitre 4 : Héritage en Java

- L'un des principes fondamentaux de la POO.
- Il nous est possible de concevoir une classe supplémentaire en copiant les attributs et les méthodes d'une classe existante.
 - ❖ La possibilité d'introduire de nouvelles propriétés et/ou méthodes est attribué à la nouvelle classe.
- Le recyclage du code est rendu plus aisé.
- Il établit une liaison entre deux classes :
 - ❖ Une classe mère, superclasse ou classe de base
 - ❖ Une classe fille, sous-classe ou classe dérivée

I. Principe

- La création d'une hiérarchie composée de superclasses et de sous-classes est rendue possible par l'enchaînement successif des héritages entre classes.
- En règle générale, les propriétés et méthodes partagées par diverses classes sont regroupées dans une seule superclasse.
- En Java, une classe :
 - ❖ L'héritage de plusieurs classes n'est pas permis.
 - ❖ Un nombre infini de classe fils.
- Toutes les classe en Java hérite de la classe **Object**.

II. Mise en œuvre

- Le terme **extend** est adopté pour indiquer La relation d'héritage entre deux classes.

```
class Fille extends Mere { ... }
```

- Le terme « super » est adopté afin de faire référence à une opération ou traiter un attribut d'une classe mère comme suit :

```
super.nom_de_la_donnee  
super.nom_de_la_methode()
```

- Le terme « super » est impliqué aussi pour appeler le constructeur de la classe parente :

```
super(param_1,param_2, ... param_n) ;
```


Exemples

```
public class Rectangle {  
  
    protected double larg;  
    protected double lng;  
  
    public Rectangle(){  
        larg=0;  
        lng=0;  
    }  
  
    public Rectangle(double larg, double lng) {  
        this.larg=larg;  
        this.lng=lng;  
    }  
  
    public double perimetre(){  
        return (larg+lng)*2;  
    }  
  
    public double surface(){  
        return larg*lng;  
    }  
}
```

```

public class Carre extends Rectangle {
    public Carre(double cote)
    {
        // appel au constructeur de la super-classe
        // "Rectangle"
        super(cote,cote);
    }

    public static void main(String[] args) {
        Carre carre = new Carre(11);

        System.out.println("Périmètre = "+ carre.perimetre());
        System.out.println("Surface = "+ carre.surface());
    }
}

```

Périmètre = 44.0
 Surface = 121.0

carre-

- larg : double - Rectangle
- lng : double - Rectangle
- clone() : Object - Object
- equals(Object obj) : boolean - Object
- finalize() : void - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- perimetre() : double - Rectangle
- surface() : double - Rectangle

Nous envisageons de redéfinir les deux méthodes car nous ne sommes pas satisfaits de l'élaboration de la super-classe.

```

public class TriangleRectangle extends Rectangle {
    public TriangleRectangle(double base, double hauteur) {
        super(base, hauteur);
    }

    public double perimetre() {
        double cote3=Math.sqrt(larg*larg+lng*lng);
        //double cote2=Math.sqrt(Math.pow(larg, 2)+Math.pow(lng, 2));
        return larg+lng+cote3;
    }

    public double surface() {
        return (larg*lng)/2;
    }
}

```

Exemple de
Polymorphisme
d'Héritage

NB : Afin de mentionner que cela concerne de méthodes redéfinies provenant d'une super-classe, il est préférable d'utiliser l'annotation `@Override` avant ces méthodes.

```
@Override
public double Surface() {
    return largeur * longueur / 2;
}

@Override
public double Perimetre() {
    return largeur + longueur + Math.sqrt(largeur*largeur+longueur*longueur);
}
```

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

    TriangleRectangle tr = new TriangleRectangle(5,10);

    Rectangle r_tr = new TriangleRectangle(1, 2);
    Rectangle r_c = new Carre(3);

    System.out.println("Périmètre tr = " + tr.perimetre());
    System.out.println("Surface tr = " + tr.surface());

    System.out.println("Périmètre r_tr= " + r_tr.perimetre());
    System.out.println("Surface r_tr = " + r_tr.surface());

    System.out.println("Périmètre r_c = " + r_c.perimetre());
    System.out.println("Surface r_c = " + r_c.surface());

}
```

```
Périmètre tr = 26.18033988749895
Surface tr = 25.0
Périmètre r_tr= 5.23606797749979
Surface r_tr = 1.0
Périmètre r_c = 12.0
Surface r_c = 9.0
```

Recommandations

- Lors de la conception d'une classe principale, il est primordial de faire preuve de prudence en ce qui concerne les éléments suivants :
 - ❖ Lors de la définition des accès aux variables d'instances, il convient de faire un choix entre les modificateurs `protected` et `private`.
 - ❖ Pour restreindre la possibilité de redéfinir une méthode, il est indispensable de la déclarer avec le modificateur `final`.
- Pendant la conception d'une classe dérivée, il est primordial de prendre en compte les différentes situations suivantes lors de l'implémentation d'une méthode héritée non finale :
 - ❖ Dans le cas où la méthode héritée est adéquate, il n'est pas obligatoire de la réinventer.
 - ❖ Lorsque la méthode héritée n'est qu'une partie du code, il est conseillé de la redéfinir en commençant par appeler la méthode héritée (via `super`) pour assurer son évolution.
 - ❖ Dans le cas où la méthode héritée ne convient pas, il est primordial de redéfinir la méthode sans faire référence à la méthode héritée.

Chapitre 5 : Classes abstraites et Interfaces en Java

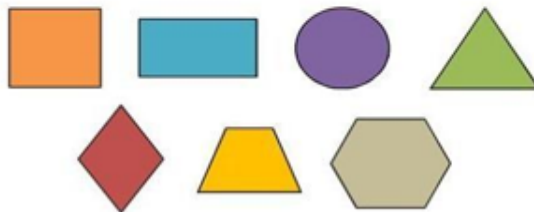
I. Usage de l'abstraction

- L'héritage offre la possibilité de regrouper divers attributs et méthodes pour un ensemble de types, CEPENDANT...
 - ❖ Par moments, il serait souhaitable de consolider un comportement (méthode) sans avoir à l'implémenter au niveau d'abstraction actuel (dans la super-classe) !!!
- L'héritage permet de regrouper certains attributs et méthodes pour plusieurs types, CEPENDANT.....
 - ❖ Par moments, il peut être souhaitable de regrouper un comportement (méthode) sans pour autant le mettre en œuvre au niveau d'abstraction actuel (dans la super-classe) !!!

Exemple :

Par exemple, un cri d'animal, le calcul de la surface d'une forme géométrique, etc.

- Nous élaborerons une (ou plusieurs) méthode(s) abstraite(s) dans la super-classe.
- Nous mettrons en œuvre ces méthodes dans les sous-classes.



1. Surface des formes géométriques

- N'est-il pas envisageable de déterminer la superficie de chaque forme géométrique, n'est-ce pas ?
- Dans la classe *FormeGeometrique*, nous ajoutons une méthode *Surface ()* sans la mettre en œuvre.
- Chaque forme a ses propres caractéristiques et sera implémentée de manière spécifique.
- Une méthode abstraite est définie comme suit :
 - ❖ Il est essentiel de précéder la méthode du mot-clé "abstract".
 - ❖ Aucun corps de méthode ne doit être spécifié.

```
public abstract double Surface();
```

- En outre, une classe sera considérée comme abstraite si elle renferme au moins une méthode abstraite.

```
public class Carre extends FormeGeographique {

    public double cote;
    @Override
    public double Surface() {

        return cote * cote;
    }

}
```

```
public abstract class FormeGeographique {

    public abstract double Surface();

}
```

```
public class Cercle extends FormeGeographique{

    public double rayon;

    @Override
    public double Surface() {

        return Math.PI * this.rayon * this.rayon;
        // return Math.PI * Math.pow(this.rayon, 2);
    }

}
```

a. Mise en œuvre de méthode abstraite

- Calculer la surface d'un carré est différent à celui d'un cercle.
 - ❖ Chaque classe doit mettre en œuvre sa propre méthode de calcul de surface.

II. Directives pour l'emploi des classes abstraites

- Lorsqu'une classe est déclarée comme abstraite, toutes les méthodes de cette classe sont automatiquement transformées en méthodes abstraites.
- Il est possible d'inclure des méthodes non abstraites dans une classe abstraite.
- L'intégralité des méthodes abstraites de la classe abstraite doivent être implémentées par les sous-classes de celle-ci.
 - ❖ Elles n'ont pas l'obligation de réimplémenter les méthodes qui ont déjà été codées dans la classe de base.
- Lorsqu'une des méthodes abstraites n'est pas concrètement implémentée, la sous-classe devient abstraite et il est nécessaire de la déclarer de manière explicite en tant que telle.
 - ❖ Étant donné qu'elle va recevoir l'héritage de quelques-unes ou la totalité des méthodes abstraites de la classe parente.
- Même si une classe abstraite possède un ou plusieurs constructeurs, il n'est pas possible de l'instancier.
 - ❖ Lorsqu'on tente d'instancier l'objet "FormeGeometrique", une erreur de type survient indiquant que "FormeGeographique est abstrait ; impossible d'être instancié".

III. Concept d'Interface

- Une autre forme d'abstraction.
 - ❖ Il est interdit d'inclure des signatures de méthodes abstraites dans une interface, à moins d'utiliser le mot clé `abstract`.
- Les interfaces fournissent une alternative à l'absence d'héritage de diverses classes.
 - ❖ Diverses interfaces sont mises en œuvre par une classe.

IV. Traits des interfaces

- Une interface peut recevoir une extension de la part d'une ou plusieurs autres interfaces.
- Les interfaces ne comportent pas de constructeurs.
- Chacune des méthodes annoncées dans une interface sont visibles publiquement.
- Il est systématique que les méthodes d'une interface soient publiques et abstraites par défaut.
- Il est nécessaire que toutes les classes qui implémentent une interface fournissent une implémentation pour l'intégralité des méthodes présentes dans cette interface.

2. Syntaxe de définition

- L'utilisation de mot clé `interface` au lieu du terme `class` dans la déclaration.
- L'ajout du mot-clé `"abstract"` n'est pas requis.
 - ❖ De manière implicite, une méthode dans une interface est définie comme publique et abstraite.
 - Ainsi, l'utilisation de `"public abstract"` n'est pas nécessaire au début de la déclaration.
 - Cependant, elles peuvent être employées si l'on juge que cela améliore la lisibilité

```
public interface MyInterface {

    void Method1( String p1 );

    String Method2();

    public abstract int Method3(int p3);

}
```

3. Syntaxe d'implémentation

- L'utilisation du terme "**implements**" est requise afin de mettre en œuvre une interface.
- Similairement à l'héritage, l'emploi de l'annotation `@Override` est conseillé
 - ❖ Cela signale que la première déclaration de la méthode se trouve dans un type parent, en l'occurrence, l'interface.

```
public class UneClasse implements MyInterface {

    @Override
    public void Method1(String p1) {
        // TODO Auto-generated method stub
    }

    @Override
    public String Method2() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int Method3(int p3) {
        // TODO Auto-generated method stub
        return 0;
    }

}
```

V. Héritage multiple

- Une classe est limitée à l'héritage d'une seule classe parente, en recourant à du terme `extends`.
- Il est maintenant possible qu'une classe hérite d'une classe de base et implémente

une interface en utilisant implements.

- ❖ Lorsque plusieurs interfaces sont implémentées, il est obligatoire de suivre la spécification d'héritage. Outre, une classe peut implémenter plusieurs interfaces en les séparant par des virgules.

Exemple d'héritage multiple grâce aux interfaces

```
public class AnotherClass extends FormeGeographique implements MyInterface, OtherInterface {  
  
    @Override  
    public double Surface() {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
    @Override  
    public void Method1(String p1) {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public String Method2() {  
        // TODO Auto-generated method stub  
        return null;  
    }  
  
    @Override  
    public int Method3(int p3) {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
    @Override  
    public void MethodOtherInterface() {  
        // TODO Auto-generated method stub  
    }  
  
}
```

Examen


Programmation Orienté Objet

Niveau : 3^{ème} Génie Mécanique et Productique

Préparé Par : HAMDI Ghassen

Docteur en Sciences Informatique.

Année universitaire: 2021/2022

République Tunisienne Ministère de l'Enseignement Supérieur et de la Recherche Scientifique Université de Sousse École Nationale d'Ingénieurs de Sousse		Formulaire	EXA-FR-02-00
		Examen	08/01/2022

Année Universitaire 2021 - 2022

Session : ☒ Principale / ☐ Rattrapage

Matière : Programmation Orienté Objet

Enseignant	: Ghassen HAMDI	Date	: 08/01/2022
Filière	: GMP	Section	: 3
Barème	: Ex. 1 : 7pts ; Ex. 2 : 6pts ; Ex. 3 : 7pts	Documents	: non autorisés
		Calculatrice	: non autorisée
Durée/Ex.	: Ex. 1 : 30mn ; Ex. 2 : 30mn ; Ex. 3 : 30mn	Durée	: 1h30mn
		Nbre de pages	: 2

N.B.

Il sera tenu compte de la lisibilité, la présentation, et de la clarté des réponses.
Les exercices sont indépendants et peuvent être traités dans n'importe quel ordre.
Numérotez vos feuilles (par exemple 1/2, 2/2 ou 1/3, 2/3, 3/3 ou ...)

Exercice 1 :

1. Ecrire une classe Java « **Exo1** » permettant d'appliquer une opération arithmétique sur deux valeurs numériques.
2. L'opérateur ainsi que les valeurs doivent être saisis à partir de la ligne de commandes selon le format suivant :

```
➤ java nom_app operateur val1 val2
```
3. Les opérateurs à considérer sont ceux de l'addition, la soustraction, la multiplication et la division.
4. Considérer ces opérateurs sous forme de caractères ou de chaînes de caractères.

Exercice 2 :

Une chaîne de caractères chaîne est **carrée** s'il existe une chaîne ch telle que chaîne=chch, par exemple chercher et bonbon sont des chaînes carrées.


Une chaîne de caractères chaîne est dite **palindrome** si elle s'écrit d'un sens à l'autre, que ce soit de gauche à droite ou de droite à gauche. A titre d'exemple, elle et radar sont des palindromes.

1. Donnez une classe Java « **Exo2** » qui nous permet de vérifier si une chaîne saisie par l'utilisateur est carrée ou palindrome.

```
➤ Pour cet exercice, développer les méthodes booléennes « EstCarrée » et « EstPalindrome ».
```

Exercice 3 :

1. Développer la classe Crayon caractérisée par :
 - deux attributs privés (double) *épaisseur* et *longueur*,
 - un constructeur qui nous permet d'initialiser les attributs
 - les mutateurs et les accesseurs nécessaires
 - une méthode **affiche()** qui nous permet d'afficher les caractéristiques d'un crayon.
2. Développer la classe CrayonCouleur héritant de la classe Crayon et qui est caractérisée aussi par :
 - un attribut supplémentaire privé *couleur* (String).
 - un constructeur qui nous permet d'initialiser les attributs
 - le mutateur et l'accesseur nécessaires
 - une méthode appelée **changeCaracteristiques()** qui n'a pas de valeur de retour et qui en faisant appel aux autres méthodes modifie les valeurs de tous les attributs (longueur, épaisseur et couleur).
 - une méthode **affiche()** qui permet de présenter les caractéristiques d'un crayon de couleur en redéfinissant et utilisant la méthode d'affichage d'un crayon ordinaire.
3. Enfin, développer la classe TestCrayon qui contient une méthode main() pour tester ces classes :
 - créer dedans un crayon et crayon de couleur, changer l'épaisseur du premier et
 - La couleur du second puis afficher leurs informations.

République Tunisienne Ministère de l'Enseignement Supérieur et de la Recherche Scientifique Université de Sousse École Nationale d'Ingénieurs de Sousse		Formulaire	EXA-FR-02-00
		Correction Examen	08/01/2022

Année Universitaire 2021 - 2022

Session : ☒ Principale / ☐ Rattrapage

Matière : Programmation Orienté Objet

Enseignant	: Ghassen HAMDI	Date	: 08/01/2022
Filière	: GMP	Section	: 3
Barème	: Ex. 1 : 7pts ; Ex. 2 : 6pts ; Ex. 3 : 7pts	Documents	: non autorisés
		Calculatrice	: non autorisée
Durée/Ex.	: Ex. 1 : 30mn ; Ex. 2 : 30mn ; Ex. 3 : 30mn	Durée	: 1h30mn
		Nbre de pages	: 2

N.B.

Il sera tenu compte de la lisibilité, la présentation, et de la clarté des réponses.
Les exercices sont indépendants et peuvent être traités dans n'importe quel ordre.
Numérotez vos feuilles (par exemple 1/2, 2/2 ou 1/3, 2/3, 3/3 ou ...)

Exercice 1 :

```
public class Exo1 {

    public static void main(String[] args) {

        // Ici, on considère l'opérateur comme un caractère
        // chose plus pratique
        char operateur = args[0].charAt(0);
        int a = Integer.parseInt(args[1]);
        int b = Integer.parseInt(args[2]);

        switch(operateur) {
            case '+' -> System.out.println("Résultat = "+ (a+b));
            case '-' -> System.out.println("Résultat = "+ (a-b));
            case 'x' -> System.out.println("Résultat = "+ a*b);
            case '/' -> System.out.println("Résultat = "+ (float)a/b);
            default -> System.out.println("Opérateur invalide");
        }

        // Ici, on considère l'opérateur comme une chaîne de caractères
        // moins pratique

        /*String operateur = args[0];
        int a = Integer.parseInt(args[1]);
        int b = Integer.parseInt(args[2]);

        switch(operateur) {
            case "plus" -> System.out.println("Résultat = "+ (a+b));
            case "moins" -> System.out.println("Résultat = "+ (a-b));
            case "fois" -> System.out.println("Résultat = "+ a*b);
            case "sur" -> System.out.println("Résultat = "+ (float)a/b);
            default -> System.out.println("Opérateur invalide");
        }*/
    }

}
```

Exercice 2 :

```
import java.util.Scanner;

public class Exo2 {

    public static boolean EstPalindrome(String mot){

        String motinverse="";
        for(int i = mot.length()-1; i>=0; i--){
            motinverse=motinverse+mot.charAt(i);
        }

        if(mot.equals(motinverse)) return true;
        else return false;

    }

    public static boolean EstCarree(String mot){

        if(mot.length()%2==0)
        {
            String moitie = mot.substring(0, (mot.length()/2));
            if(mot.equals(moitie+moitie)) return true;
        }
        return false;

    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner( System.in );
        System.out.println("Donnez une chaine de caractères :");
        String mot = scanner.next();
        if(EstCarree(mot)) System.out.println("Ce mot est carré...");
        else if(EstPalindrome(mot)) System.out.println("Ce mot est palindrome...");
        else System.out.println("Ce mot n'est ni carré ni palindrome !!!");

        scanner.close();

    }
}
```

Exercice 3 :

```

public class CrayonCouleur extends Crayon {

    private String couleur;

    public CrayonCouleur(double epaisseur, double longueur, String couleur) {
        super(epaisseur, longueur);
        this.couleur=couleur;
    }

    public String getCouleur() {
        return couleur;
    }

    public void setCouleur(String coul) {
        couleur=coul;
    }

    public void changeCaracteristiques(double newEp, double newLong, String newCoul) {
        setEpaisseur(newEp);
        setLongueur(newLong);
        setCouleur(newCoul);
    }

    public void affiche() {
        super.affiche();
        System.out.println("Couleur = "+ couleur);
    }
}

public class Crayon {

    private double epaisseur;
    private double longueur;

    public Crayon(double epaisseur, double longueur) {
        this.epaisseur=epaisseur;
        this.longueur=longueur;
    }

    public double getEpaisseur() {
        return epaisseur;
    }

    public double getLongueur() {
        return epaisseur;
    }

    public void setEpaisseur(double ep) {
        epaisseur=ep;
    }

    public void setLongueur(double lng) {
        longueur=lng;
    }

    public void affiche() {
        System.out.println("Epaisseur = "+epaisseur + " ; Longueur = "+ longueur);
    }
}

```

```
public class TestCrayons {  
    public static void main(String[] args) {  
        Crayon cr = new Crayon(1, 10);  
        CrayonCouleur crc1 = new CrayonCouleur(1.2, 11, "jaune");  
        CrayonCouleur crc2 = new CrayonCouleur(0.8, 9, "rouge");  
  
        cr.affiche();  
        crc1.affiche();  
        crc2.affiche();  
  
        cr.setEpaisseur(0.9);  
        crc1.setCouleur("mauve");  
        crc2.changeCaracteristiques(0.5, 7, "marron");  
  
        cr.affiche();  
        crc1.affiche();  
        crc2.affiche();  
    }  
}
```