Lexique

Variable : Une variable est un emplacement de stockage qui porte un nom unique et qui peut contenir des données (ou résultats) pouvant être modifiés pendant l'exécution du programme. Les variables sont emmagasinées dans la mémoire de l'ordinateur. Une variable a quatre propriétés principales:

- Nom : utilisé pour y faire référence
- Contenu: valeur dans la boîte d'information de la variable
- Type: spécifie le type de données pouvant être conservé dans la variable
- Adresse: localisation physique dans la mémoire RAM

Conventions de nommage :

- Le nom d'une variable est représenté par un identificateur d'une longueur illimitée.
- Le nom d'une variable est sensible à la casse (majuscules, minuscules)
- Certains mots-clés sont réservés par le langage et ne peuvent être utilisés comme nom de variables (exemple : class, byte, case, int, char, etc.)
- L'espace ou le caractère de fin de ligne sépare les identificateurs entre eux.
- Le nom des variables est écrit en minuscules sauf s'il comporte plusieurs mots, dans ce cas la lettre de chaque mot (sauf le premier mot) est écrit en MAJUSCULE. Évidemment il n'y a pas d'espaces dans un nom de variable
- On préconise l'utilisation de mots complets plutôt que des abréviations pour identifier des variables, cela permet d'auto-documenter le code.

Une variable peut être utilisée de plusieurs façons :

- **Variables d'instance (attributs) :** ce sont elles qui définiront les caractéristiques de notre objet.
- **Variables de classe :** communes à toutes les instances de votre classe.
- **Variables locales :** variables utilisées pour travailler dans l'objet.
- **Constantes :** utilisée pour contenir un élément qui ne changera pas pendant l'exécution du programme
- **Paramètre :** valeur envoyée à une variable déclarée dans la signature d'une méthode

Méthodes :
Constructeurs : méthodes servant à créer des objets
Accesseurs : Un accesseur est une méthode qui va permettre d'accéder aux variables d'instances de nos objets en lecture (rappelez-vous, les variables d'instance sont privées). Ces méthodes vont donc retourner un résultat, la valeur de la variable, mais ne reçoivent aucun paramètre, puisqu'elles ne modifient rien.
Mutateurs : Un mutateur nous permet de faire l'inverse de l'accesseur, soit d'accéder aux variables d'instance en écriture, donc pouvoir modifier le contenu de la variable. Ces méthodes vont donc recevoir une valeur en paramètre de façon à pouvoir modifier la donnée. Ces méthodes retournent un boolean indiquant si la modification a réussi ou ne retournent rien (si pas de validation).
Main : Une seule méthode main peut être active par projet. C'est le point de départ du programme.
Méthodes d'instance : méthodes servant à la gestion des objets.
Les méthodes d'instance sont déclarées de la manière suivante :

```
<type d'accès> <type de retour> <nom de la méthode>(<paramètres d'entrée)
public void afficherValeur(String pMessage)
private int recupererValeur()
```

This : Le mot-clé this sert à faire référence à l'objet en cours d'utilisation. Il ne peut être utilisé qu'à l'intérieur d'une méthode et constitue une référence sur l'objet pour lequel cette méthode a été appelée. On peut utiliser cette référence comme toute autre référence vers un objet. Il sert souvent à lever l'ambiguïté dans le code.

énoncé **d'importation**: sert à utiliser les classes d'un autre package

```
import java.util.Scanner;
```

déclaration d'une classe: permet de définir une nouvelle classe

```
public class Carte
{
```

attributs ou **variables d'instances**

visibilité: détermine comment les autres classes voient cet élément

```
    private String couleur;
    private int    valeur;
```

paramètre: comme une variable locale mais permet de recevoir des données

constructeur: permet de créer un objet avec des données initiales valables

```
    public Carte (int pvaleur, String pcouleur)
    {
        this.valeur = pvaleur;
        this.setValeur(pvaleur);
        this.setCouleur(pcouleur);
    }
```

affectation: met une valeur dans une variable

appel de méthode: une méthode peut appeler une autre méthode. Si l'objet n'est pas spécifié c'est le this, qui représente l'objet appelant, qui est implicitement impliqué.

instruction: unité d'exécution

mutateur: méthode qui a pour but de modifier un attribut

```
    public void setCouleur(String pcouleur)
    {
        couleur = pcouleur;
    }
```

accesseur: méthode qui a pour but de consulter un attribut

```
    public int getValeur()
    {
        return valeur;
    }
```

main: méthode principale par lequel démarre toujours un programme

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

commentaires: permet de donner des informations supplémentaires aux programmeurs qui consulte le code

```
        // Création des cartes
        Carte carte1;
        carte1 = new Carte(1, "coeur");
        int a = 5;
    }
}
```

déclaration d'une variable Objet : dit au compilateur de réserver la mémoire nécessaire pour une variable objet (64bits)

opérateur new : nécessaire pour construire un objet: alloué la mémoire nécessaire et appeler le bon constructeur

création d'une **instance** ou d'un **objet**: (se dit aussi instanciation) dit au compilateur de réserver la mémoire nécessaire pour tenir un objet puis appelle le constructeur de l'objet en question

déclaration d'une **variable** de **type de base** et affectation d'une valeur : dit au compilateur de réserver la mémoire nécessaire pour retenir un entier

déclaration d'un variable locale: réserve de la mémoire pour le temps où s'exéc12

Entrées/Sorties

Description	Exemple d'utilisation

Récupération d'un élément à partir d'une boîte de dialogue (Swing)	`String reponse;`
	`reponse = JOptionPane.showInputDialog(null,"Quel est ton nom ?");`
	`System.out.println(reponse);`

Conversion des données reçues (Swing)	`Etudiant maurice = new Etudiant("Tremblay", "Maurice", 18, "Réseau");`
	`String rep = JOptionPane.showInputDialog("Age");`
	`maurice.setAge(Integer.parseInt(rep));`
	`System.out.println(maurice.getAge());`

Affichage d'une boîte de message	`JOptionPane.showMessageDialog(null, "L'aire du rectangle est: " + parAire + " cm carrés" + "\nLe périmètre du rectangle est: " + parPerimetre + " cm", "Résultat", .INFORMATION_MESSAGE);`
Récupération des données à partir d'un Scanne	`Scanner entree = new Scanner(System.in);` `base = entree.nextInt();`
Affichage de données dans la console	`System.out.println("Bonjour");`

Priorité des opérateurs

Priorité	Opérateur
1	() []
	++ -- !
2	* / %
3	+ -
4	< > <= >=
5	== !=
6	&&
7	||
8	= += -= *= /= %= &= ^= |= <<= >>=
9	,

Types de données primitifs

Type de donnée	Type élémentaire	Description
Entiers	**byte**	1 octet -128,+127
	short	2 octets -32768, +32767
	int	4 octets -2147438648, +2147483647
	long	8 octets -9233372036854775808, …

Flottants	**float**	4 octets $-1.4^{E}-45$, $3.4^{E}38$ (précision 7)
		Par défaut, les variables sont créées en double, pour forcer float il faut faire suivre de la lettre f ou F.
		float x;
		x = 12.5; //erreur
		x = 12.5f; //ok

	double	8 octets $4.94^{E}-394$, $1.79^{E}308$ (précision 15)
Caractère	**char**	2 octets 'c' (unicode)
Booléen	**boolean**	1 octet true et false

Types complexes

Description	Utilisation
String	Permet de contenir une chaîne de caractères `String <nom de la variable> = « valeur de la String »;`

Objet	Permet de contenir un objet avec des attributs définis par l'utilisateur
	`<type de l'objet> <nom de la variable> = new <type de l'objet>();`
	
	`<type de l'objet> <nom de la variable> = new <type de l'objet>(valeurs initialisées selon le constructeur);`

Tableaux statique vs dynamique

Description	Tableau statique	Tableau dynamique

Déclaration	`<type du tableau> <nom du tableau> [] = new <type du tableau> [taille];`
	
	`<type du tableau> <nom du tableau> [] = { <contenu du tableau>};`
		`ArrayList <<type>> <nomListe> = new ArrayList();`

Taille	`tableau.length`	`tableau.size()`
Récupérer élément	`tableau[position]`	`tableau.get(position)`
Modifier un élément	`tableau[position] = nouvel élément`	`tableau.set(position, nouvel élément)`

Ajouter un élément	N/A	`tableau.add(nouvel élément)`
		Ajouter à une position précise :
		`tableau.add(position, élément)`

Supprimer un élément	Peut être mis à *null* mais espace mémoire demeure réservé	`tableau.remove(position)`
Vider un tableau	Doit faire une boucle pour mettre à *null* mais espace mémoire demeure réservé	`tableau.clear()`

Parcourir un tableau	`for (int i = 0; i < tableau.length ; i ++)`
	`{`
	`  System.out.println(tableau[i]+ " ");`
	`}`
		`for(int i=0; i < groupe.size(); i++)`
		`{`
		`  System.out.println(groupe.get(i));`
		`}`

Types d'opérateurs

Opérateurs arithmétiques		
+	addition	2 + 2 = 4
-	soustraction	3 – 2 = 1
*	multiplication	3 * 2 = 6
/	division	4 / 2 = 2
%	Modulo (reste de la division)	16 % 3 = 1
Opérateurs d'assignation		
=	affectation usuelle	a = 2 + 2 a = 4
+=	additionne deux valeurs et stocke le résultat dans la variable (à gauche)	a = 4 a += 3 identique à a = a + 3 a = 7
-=	soustrait deux valeurs et stocke le résultat dans la variable	a = 4 a -= 3 identique à a = a – 3 a = 1
***=**	multiplie deux valeurs et stocke le résultat dans la variable	a = 3 a *= 2 a = 6
/=	divise deux valeurs et stocke le quotient dans la variable	a = 4 a /= 2 a = 2
%=	divise deux valeurs et stocke le reste de la division dans la variable	a = 4 a %= 3 a = 1
Opérateurs d'incrémentation et de décrémentation (unaire)		
++	incrément	a++ (ajoute 1 à a)
--	décrément	a-- (soustrait 1 de a)
Opérateurs relationnels		
==	égal	4 == 2 + 2 (vrai)
!=	différent	4 != 2 + 2 (faux)
>	plus grand	4 > 2 + 2 (faux)
>=	plus grand ou égal	4 >= 2 + 2 (vrai)
<	plus petit	4 < 2 + 2 (faux)
<=	plus petit ou égal	4 <= 2 + 2 (vrai)
Opérateurs logiques		
&&	et	true && true = true true && false = false false && false = false false && true = false
||	ou	true || false = true false || false = false false || true = true true || true = true
!	Non logique	!true = false !false = true

Instructions conditionnelles

Non / Description	Utilisation	Exemple

if / SI
Structure de base permettant d'exécuter une série d'instructions lorsqu'une condition est réalisée
	`if (condition)`
	`{`
	`  //liste d'instructions si la`
	`  //condition est vraie`
	`}`
	`else`
	`{`
	`  //liste d'instructions si la`
	`  //condition est fausse`
	`}`
		`boolean reponse = false;`
		`if (nombre % 2 == 0)`
		`{`
		`  reponse = true;`
		`}`

if-else-if / SI-SINON-SI
Permet d'exécuter des instructions si une condition est vérifiée et qu'on veut exécuter un seul de ces blocs d'instructions
	`if (condition 1)`
	`{`
	`  //liste d'instructions si la`
	`  //condition 1 est vraie`
	`}`
	`else if (condition 2)`
	`{`
	`  //liste d'instructions si la`
	`  //condition 2 est vraie`
	`}`
	`else if (condition n)`
	`{`
	`  //liste d'instructions si la`
	`  //condition n est vraie`
	`}`
	`else`
	`{`
	`  //liste d'instructions si aucune`
	`  //condition n'est vraie`
	`}`
		`if (note >= 90)`
		`{`
		`  grade = 'A';`
		`}`
		`else if (note >= 80)`
		`{`
		`  grade = 'B';`
		`}`
		`else if (note >= 70)`
		`{`
		`  grade = 'C';`
		`}`
		`else if (note >= 60)`
		`{`
		`  grade = 'D';`
		`}`
		`else if (note >= 50)`
		`{`
		`  grade = 'E';`
		`}`
		`else`
		`{`
		`  grade = 'F';`
		`}`

Condition complexe
Permet de combiner diverses conditions et d'exécuter les instructions en fonction de la combinaison booléenne des conditions
	`if(!condition)…`
	`if ((condition1)&&(condition2))…`
	`if ((condition1)||(condition2))…`
		`if (note >= 90 && note <= 100)`
		`{`
		`  grade = 'A';`
		`}`
		`else if (note >= 80 && note < 90)`
		`{`
		`…`

switch / SELON LE CAS
Permet de faire plusieurs tests sur le contenu d'une seule variable. Fonctionne uniquement sur les types byte, short, char, int et String.
	`switch (variable)`
	`{`
	`  case valeur1 :`
	`  //Liste d'instructions`
	`  break;`
	
	`  case valeur2 :`
	`  //Liste d'instructions`
	`  break;`
	
	`  case valeurN :`
	`  //Liste d'instructions`
	`  break;`
	
	`  default:`
	`  //Liste d'instructions`
	`}`
		`switch (jour)`
		`{`
		`  case 1:`
		`  jourSemaine = "Dimanche";`
		`  break;`
		`  case 2:`
		`  jourSemaine = "Lundi";`
		`  break;`
		`  case 3:`
		`  jourSemaine = "Mardi";`
		`  break;`
		`  case 4:`
		`  jourSemaine = "Mercredi";`
		`  break;`
		`  case 5:`
		`  jourSemaine = "Jeudi";`
		`  break;`
		`  case 6:`
		`  jourSemaine = "Vendredi";`
		`  break;`
		`  case 7:`
		`  jourSemaine = "Samedi";`
		`  break;`
		`  default:`
		`  jourSemaine = "Invalide";`
		`}`

Instructions répétitives

Itérations : Nombre de fois que la boucle est exécutée (Une itération = un passage dans la boucle)

Non / Description	Utilisation	Exemple

while (TANT QUE)
Permet d'exécuter plusieurs fois la même d'instructions (tant que la condition est vraie).
	`while (condition)`
	`{`
	`  //liste d'instructions`
	`}`
		`int fois = 0;`
		`while (fois <= 10)`
		`{`
		`  System.out.println("Voici la ligne " + fois);`
		`  fois++;`
		`}`

do while (FAIRE … TANT QUE)
Variante de la boucle *while* permettant d'exécuter au moins une fois les instructions car la condition est vérifiée à la fin
	`do`
	`{`
	`  liste d'instructions`
	`}`
	`while (condition réalisée);`
		`int fois = 0;`
		`do`
		`{`
		`  System.out.println("Voici la ligne " + fois);`
		`  fois++;`
		`}`
		`while (fois <= 10);`

for (POUR)
Permet de répéter la même série d'instructions un nombre de fois défini.
	`for (compteur; condition; modification du compteur)`
	`{`
	`liste d'instructions`
	`}`
		`for(int i = 1; i <= 10; i++)`
		`{`
		`  System.out.println("Voici la ligne " + i);`
		`}`

Boucles avec plusieurs conditions
Permet de vérifier des conditions complexes mettant en relation des opérateurs logiques
	`while (fois >= 0 && fois<= 10)`
	`{`
	`  System.out.println("Voici la ligne " + fois);`
	`  fois++;`
	`}`
		`while (fois > 10 || fois == 0)`
		`{`
		`  System.out.println("Voici la ligne " + fois);`
		`  fois++;`
		`}`

Boucles imbriquées
Boucle dans une boucle, peut se faire avec n'importe quelle structure répétitive
	`for (int i = 0; i < 5; i++)`
	`{`
	`  for (int j = 0; j < 3; j++)`
	`  {`
	`  System.out.print('*');`
	`  }`
	`  System.out.println();`
	`}`
		`for (int i = 0; i < 5; i++)`
		`{`
		`  for (int j = 0; j < 3; j++)`
		`  {`
		`  System.out.print('*');`
		`  }`
		`  System.out.println();`
		`}`

Boucle for-each
Permet d'itérer sur une collection d'éléments (Ex : ArrayList)
	`for (<type> objet : <ArrayList>)`	`for(Etudiant etudiant : groupe)`
		`{`
		`  System.out.println(etudiant);`
		`}`

Validations

Il est préférable d'isoler le code de validation dans des méthodes distinctes:

- Pour ne pas encombrer le code à valider
- Pour faciliter la compréhension du code
- Pour réutiliser les tests lorsque ceux-ci peuvent être appelés à plusieurs endroits dans la classe (ex**: constructeur et mutateur)**

Type	Validations à faire
int, short, long	Si l'on reçoit un paramètre "pEntier" on peut valider: pEntier > ou < ou == 0 pEntier < ou > valeur limite pEntier != Integer.MaxValue ou != Integer.MinValue
float	Si l'on reçoit un paramètre "pNombre" on peut valider: pNombre > ou < 0 pNombre < ou > valeur limite Math.abs(VALEUR_ATTENDUE – pNombre) > DELTA (DELTA est une petite valeur ex: 0,00001) Que le nombre n'est pas Float.NEGATIVE_INFINITY Que le nombre n'est pas Float.POSITIVE_INFINITY Que le nombre n'est pas Float.NaN
char	Est dans un ensemble de caractères attendus (a..z, A..Z, 0..9…).
boolean	Aucune validation
String	Comme tout objet, la chaîne de caractères ne peut pas être nulle. pChaineDeCaractère != null; En plus, on peut vouloir valider sa taille et son contenu. pChaineDeCaractère.length() < NOMBRE-MAX-CARACTERES pChaineDeCaractère.length() > NOMBRE-MIN-CARACTERES
Objet	Pas null Toute autre validation jugée pertinente
Tableaux	Étant donné que c'est un objet, pas null pTableau != null (obligatoire) Valider que le tableau n'est pas vide (optionel) pTableau.size != 0; Valider que le tableau respecte les dimensions attendues(optionel) pTableau.size < VALEUR_ATTENDUE pTableau.size > VALEUR_ATTENDUE