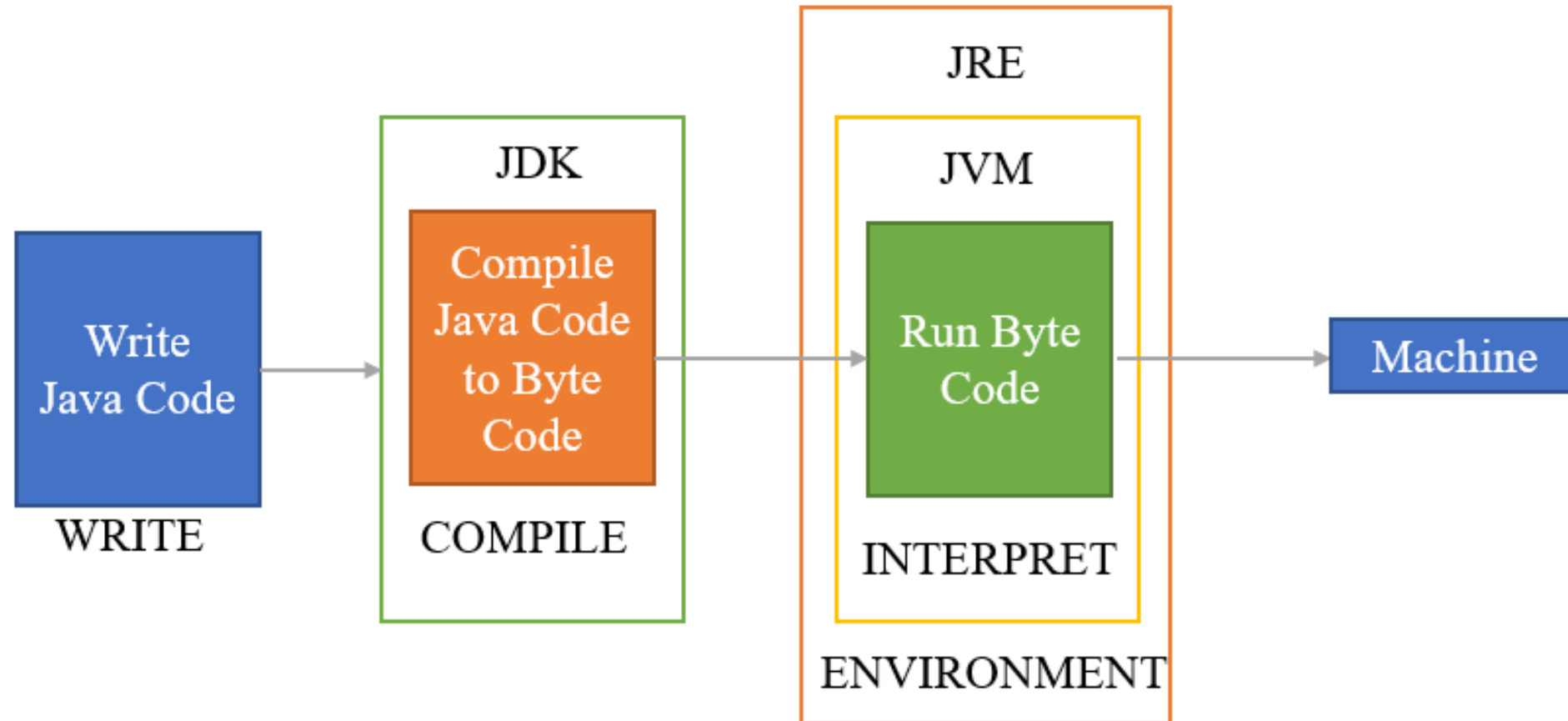


JAVA

Cour 1

Java



- Les programmes Java sont structurés en **packages** et en **classes**.
- Aucun code n'est écrit en dehors d'une classe, ce qui signifie que **toutes les fonctions sont des méthodes** en Java.
- Les packages sont mappés dans des **dossiers** et les classes dans des **fichiers**.
- La commande **javac** convertit le code Java en **Bytecode**.
- La commande **java** exécute le programme actuel en exécutant la fonction `main` dans la classe fournie.

Écrivez une fonction

- Il existe trois types de classes :

Les **classes modèles** qui sont utilisées comme modèles pour l'instanciation des objets.

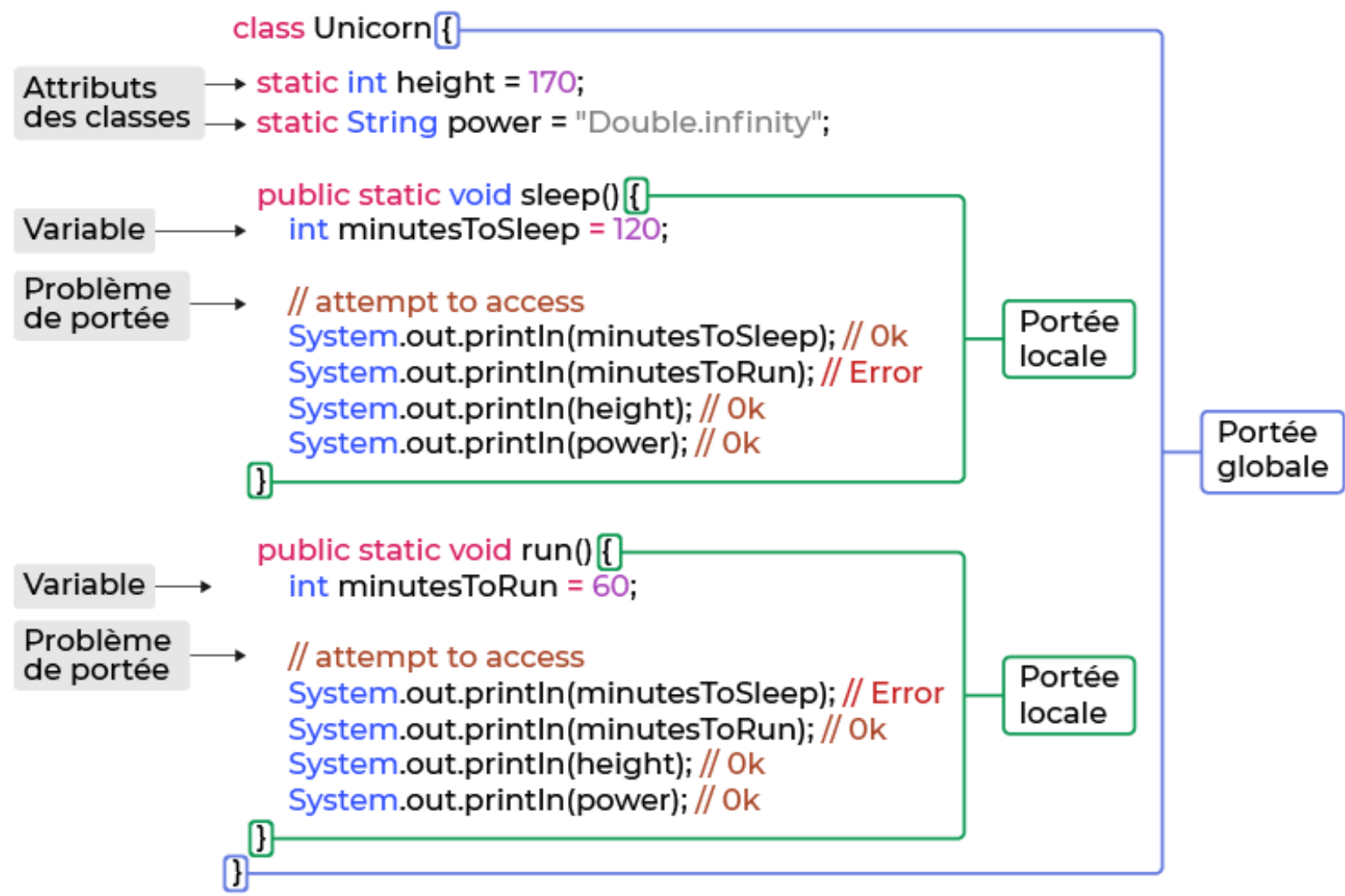
Les **classes utilitaires** qui contiennent des méthodes statiques qui peuvent être appelées directement sur la classe.

Les **classes services** qui contiennent des méthodes ou des services

Écrivez une fonction

- Vous pouvez accompagner vos classes et méthodes avec des commentaires de documentation, écrits entre `/**` et `*/`, pour générer une page HTML avec toute la documentation de la classe, appelée un **Javadoc**.
- La méthode `main` peut vous être **masquée** si vous utilisez un **framework**.
- Les principes du code propre exigent qu'**aucune logique ne soit écrite à l'intérieur de la méthode** `main`. Tout le travail doit être délégué à des fonctions bien nommées.

Portée des variables



Portée des variables

- **public** : visible pour tous et par conséquent le moins restrictif ;
- **protected (protégé)** : visible pour le package et l'ensemble de ses sous-classes ;
- **package-protected (protégé par paquet)** : généralement visible uniquement par le package dans lequel il se trouve (paramètres par défaut). Ne pas mettre de mot clé déclenche ce niveau de contrôle ;
- **private (privé)** : accessible uniquement dans le contexte dans lequel les variables sont définies (à l'intérieur de la classe dans laquelle il est situé).

DÉCLARATION DE VARIABLES

```
public class Test {  
    public static void main(String[] args) {  
        byte b = 127;  
        short s = 32767;  
        int i = 999999999;  
        long l = 999999999999999999999999L;  
        float f = 1.0E37f;  
        double d = 1.0E308;  
        boolean bl = true;  
        System.out.println(bl);  
    }  
}
```


DÉCLARATION DE VARIABLES

☐ **Déclarer des variables pour sauvegarder les valeurs suivants:**

Valeurs	Valeurs
120	999999999999999999999998
9999999999	2.0E37f
32767	2.0E308

biyt

int

Short

Long

float

double

LES OPÉRATEURS D'ASSIGNATION ET D'INCRÉMENTATION

Exemple	Résultat (avec x valant 7)
$x+3$	10
$x-3$	4
$x*3$	21
$x/3$	2.3333333
$x=3$	Met la valeur 3 dans la variable x

Opérateur	Effet
$+=$	addition deux valeurs et stocke le résultat dans la variable (à gauche)
$-=$	soustrait deux valeurs et stocke le résultat dans la variable
$*=$	multiplie deux valeurs et stocke le résultat dans la variable
$/=$	divise deux valeurs et stocke le résultat dans la variable

LES OPÉRATEURS D'ASSIGNATION ET D'INCRÉMENTATION

Opérateur	Dénomination	Effet
++	Incrémentation	Augmente d'une unité la variable
--	Décrémentation	Diminue d'une unité la variable

LES OPÉRATEURS DE COMPARAISON

Opérateur	Dénomination	Effet
== A ne pas confondre avec le signe d'affectation (=)!!	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur

LES OPÉRATEURS LOGIQUES (BOOLÉENS)

Opérateur	Dénomination	Effet
	OU logique	Vérifie qu'une des conditions est réalisée
&&	ET logique	Vérifie que toutes les conditions sont réalisées
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur <i>True</i> si la variable vaut <i>False</i> , <i>False</i> si elle vaut <i>True</i>)

LES OPÉRATEURS BIT-À-BIT

Opérateur	Dénomination	Effet
&	ET bit-à-bit	Retourne 1 si les deux bits de même poids sont à 1
	OU inclusif	Retourne 1 si l'un ou l'autre des deux bits de même poids est à 1 (ou les deux)
^	OU exclusif	Retourne 1 si l'un des deux bits de même poids est à 1 (mais pas les deux)

Syntaxe	Résultat
9 & 12 (1001 & 1100)	8 (1000)
9 12 (1001 1100)	13 (1101)
9 ^ 12 (1001 ^ 1100)	5 (0101)

LECTURE/ÉCRITURE DANS UNE VARIABLE

```
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        byte b = sc.nextByte(); System.out.println(b);
        short s = sc.nextShort(); System.out.println(s);
        int i = sc.nextInt(); System.out.println(i);
        long l = sc.nextLong(); System.out.println(l);
        float f = sc.nextFloat(); System.out.println(f);
        double d = sc.nextDouble(); System.out.println(d);
        boolean bl = sc.nextBoolean(); System.out.println(bl);
    }
}
```

Utilisation d'un nouveau objet

TRAITEMENT CONDITIONNEL

- L'instruction if ... Else if:

```
public class Test {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        short x = sc.nextShort();  
        if (x > 0) {  
            System.out.println(x + " est strictement positif");  
        } else if (x == 0) {  
            System.out.println(x + " est nul");  
        } else {  
            System.out.println(x + " est strictement négatif");  
        }  
    }  
}
```

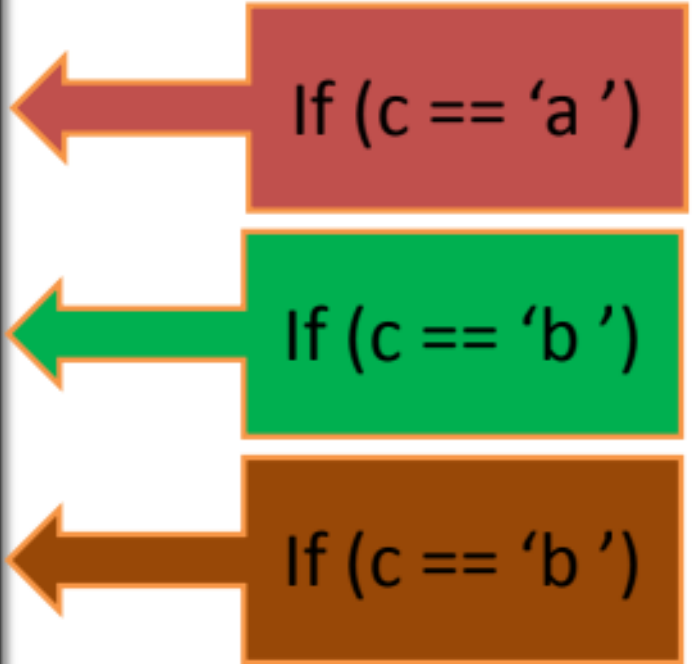

OPÉRATEUR TERNAIRE

❑ **(condition) ? instruction si vrai : instruction si faux;**

```
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        short x = sc.nextShort();
        boolean b = (x>10)? true:false;
        System.out.println(b);
    }
}
```

CHOIX MULTIPLES: SWITCH

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    char c = sc.nextLine().charAt(0);  
    switch (c) {  
        case 'a':  
            System.out.println("Je suis A");  
            break;  
        case 'b':  
            System.out.println("Je suis B");  
            break;  
        case 'c':  
            System.out.println("Je suis C");  
            break;  
        default:  
            System.out.println("Je suis "+c);  
            break;  
    }  
}
```



If (c == 'a ')

If (c == 'b ')

If (c == 'b ')

INSTRUCTION RÉPÉTITIVE / WHILE

```
public class InstructionRepetitive {  
    public static void main(String[] args) {  
        int compteur = 1;  
        while (compteur <= 5)  
        {  
            System.out.println (compteur);  
            compteur++;  
        }  
        System.out.println ("Done");  
    }  
}
```

INSTRUCTION RÉPÉTITIVE / BOUCLE IMBRIQUÉE

```
public class InstructionRepetitive {  
    public static void main(String[] args) {  
        int compteur = 1;  
        while (compteur <= 5)  
        {  
            int i = 0;  
            while( i <= 3) {  
                System.out.print (compteur*i);  
                i++;  
            }  
            compteur++;  
            System.out.println ();  
        }  
        System.out.println ("Done");  
    }  
}
```

17/03/2022 T.HAJJI-POO

INSTRUCTION RÉPÉTITIVE / DO

```
public class InstructionRepetitive {  
    public static void main(String[] args) {  
1      int compteur = 1;  
2      do {  
3          System.out.println(compteur);  
4          compteur++;  
5      } while (compteur <= 5);  
      System.out.println("Done");  
    }  
}
```

INSTRUCTION RÉPÉTITIVE / FOR

```
public class InstructionRepetitive {  
    public static void main(String[] args) {  
        for (int compteur = 1; compteur <= 5; compteur++) {  
            System.out.println(compteur);  
        }  
        System.out.println("Done");  
    }  
}
```

1
2
3
4
5
Done

INSTRUCTION CONTINUE

```
public class InstructionRepetitive {  
    public static void main(S  
        int cnt = -3;  
        while (cnt < 2) {  
            cnt++;  
            if (cnt == 0) {  
                continue;  
            }  
            System.out.println  
        }  
    }  
}
```

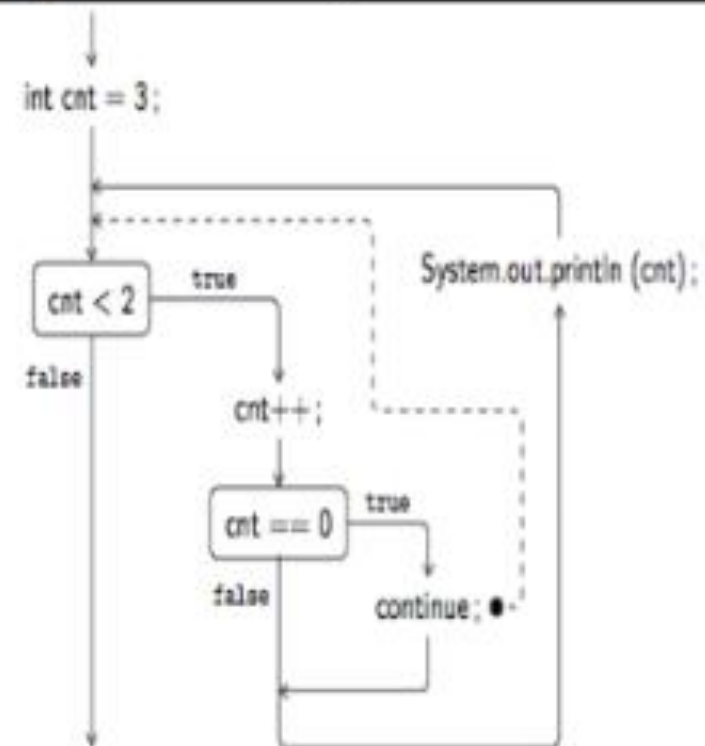


Figure 20. Déroulement de l'exécution d'une boucle while contenant une instruction continue

INSTRUCTION CONTINUE

```
public class InstructionRepetitive {  
    public static void main(String[] args) {  
        int i = 0; int j = 0;  
        externe: while (i < 2) // Boucle externe  
        {  
            interne: while (j >= 0) // Boucle interne  
            {  
                System.out.println(i+" , "+j);  
                j++;  
                if (j > 2) {  
                    break externe;  
                }  
            }  
            i++; j = 0;  
        }  
    }  
}
```

0	,	0
0	,	1
0	,	2