storre

A Multipurpose PowerPoint Template

Objectifs du cours

Dans ce cours, vous apprendrez à implémenter la logique d'application en utilisant Java SE :

- Décrire l'approche de la programmation orientée objet ;
- Expliquer la syntaxe et les conventions de codage en Java ;
- Utiliser les structures et opérateurs Java ;
- Utiliser les API de base de Java telles que les collections, les flux (streams) et les flux d'entrée/sortie (IO Streams) ;
- Déployer des applications Java SE.

Programme

I Introduction & structure

- **Environnement Java** : compilation & interprétation, JDK, JVM, JRE
- **Structure d'une classe Java** : attributs, méthodes, classes et fichier source, écriture d'une fonction main
- **Packages** : déclaration, importation
- **Création des objets :** constructeurs, lectures & écriture des attributs, exécution des instances, ordre d'initialisation
- **Variables :** types de données, déclaration des variables, initialisations des variables, suppression des variables
- **Opérateurs :** types, affectation des valeurs, caste
- **Méthodes :** design d'une méthode, déclaration des variables locales et d'instance, Varargs, accès au données statiques, passage des paramètres, surcharge des méthodes (overloading)

II Programmation O. O. avec Java

- **Héritage** : déclaration des sous-classes, héritage unique vs l'héritage multiple, déclarations des constructeurs, initialisation des objets, héritage des attributs
- **Classes abstrait:** création, méthodes abstraits, utilisation
- **Classes immutables**
- **Interfaces :** déclaration, héritage, méthodes, implémentations des interfaces
- **Enum :** déclaration, utilisation dans switch, insertion des constructeurs, méthodes et attributs
- Polymorphisme : objet vs référence, caste des objets, polymorphisme vs overiding, overiding vs hiding attributs
- **Tableau** : déclaration, passage des paramètres, accès, utilisation
- **Collections** : list, set, queu, map, trie des données

III Exceptions

- **Introduction au concept :**types des exceptions, throw & throws, méthodes(appel, redéfinition), affichage d'une exception
- **Les classes d'exception**
- **Gestion des exceptions**
- **Exceptions personnalisées**

IV Expressions LAMBDA

- **Ecriture d'une expression lambda**
- **Coder une interface fonctionnelle**
- **Les références des méthodes**
- **Utilisation des interfaces fonctionnelles intégrées**

Programme

V Collections Vs streams

- **Obtaining a Stream From a Collection**
- **Stream Processing Phases**
- **Stream.filter()**
- **Stream.map()**
- **Stream.collect()**
- **Stream.min() and Stream.max()**
- **Stream.count()**
- **Stream.reduce()**

VI Entrées / sorties

- **Référencements des fichiers & répertoires**
- **Introduction au I/O streams**
- **Lecture & écriture**
- **Sérialisation des données**
- **Interaction avec l'utilisateur**

VII Généricité

- **Introduction à la généricité en Java**
- **Apprendre à coder une classe générique**
- **Définition de méthodes génériques**

VIII Interfaces graphiques

- **introduction,**
- **Création de fenêtres et composants de base :** Jframe, Jpanel, JButton (bouton), JLabel (étiquette), JTextField (champ de texte), JTextArea (zone de texte)
- **Disposition des composants (Layouts) :** FlowLayout, BorderLayout ,GridLayout
- **Gestion des événements**
- **Composants avancés et boîtes de dialogue**
- **Gestion de l'apparence et personnalisation**
- **Accès à une base de données**

Chapitre I

Introduction & structure

1 - Environnement Java

Introduction

La différence entre un langage O. O. et un langage procédurale ?

Programmation Orientée Objet (POO)

- **Concepts Clés** : Objets, classes, héritage, encapsulation, polymorphisme.
- **Structure** : Le code est organisé autour d'objets, qui sont des instances de classes. Chaque objet contient des données (attributs) et des méthodes (fonctions) pour manipuler ces données.
- **Avantages** : Favorise la réutilisabilité du code, la modularité et la facilité de maintenance. Les objets peuvent être utilisés dans différents programmes sans modification.
- **Exemples** de Langages : Java, C++, Python.

Programmation Procédurale

- **Concepts Clés** : Procédures (ou fonctions), variables, structures de contrôle.
- **Structure** : Le code est organisé en une série de procédures ou de fonctions qui exécutent des opérations sur des données. Les données et les fonctions sont séparées.
- **Avantages** : Simple à comprendre et à utiliser pour des tâches linéaires ou séquentielles. Efficace pour les petits programmes ou les scripts.
- **Exemples de Langages** : C, Pascal, Fortran.

Chap. I

1 - Environnement Java

Introduction

Les quartes principes de la programmation orienté objet sont : abstraction, encapsulation, héritage, polymorphisme

Abstraction

Il permet de créer une image simplifié et pertinentes des objets dans le monde réel mais selon le contexte du problème et de la solution proposé.
Ses avantages :

- ✓ Gestion de la complexité
- ✓ Extensibilité
- ✓ Contrat pour les développeurs

```java
// Classe abstraite qui définit un concept général d'animal
abstract class Animal {
    // Méthode abstraite, sans implémentation,
    //à définir dans les sous-classes
    abstract void emettreSon();
}

// Classe Chien qui hérite d'Animal et implémente la méthode
class Chien extends Animal {
    @Override
    void emettreSon() {
        System.out.println("Le chien aboie.");
    }
}

// Classe Oiseau qui hérite d'Animal et implémente la méthode
class Oiseau extends Animal {
    @Override
    void emettreSon() {
        System.out.println("L'oiseau chante.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal monChien = new Chien();
        Animal monOiseau = new Oiseau();

        monChien.emettreSon();  // Affiche : Le chien aboie.
        monOiseau.emettreSon(); // Affiche : L'oiseau chante.
    }
}
```

1 - Environnement Java

Introduction

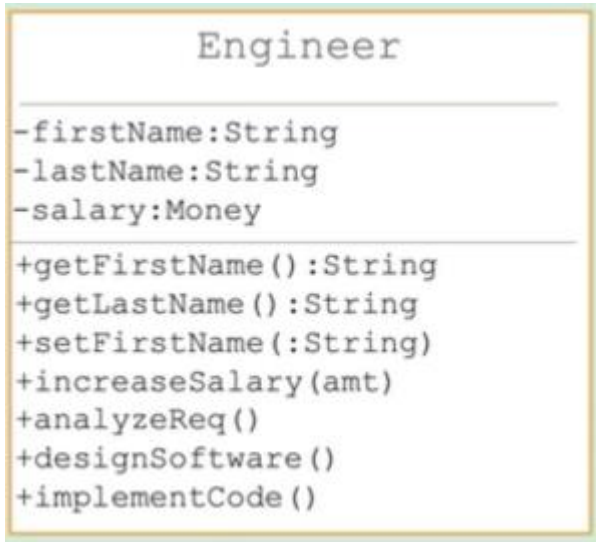
Les quartes principes de la programmation orienté objet sont : abstraction, encapsulation, héritage, polymorphisme

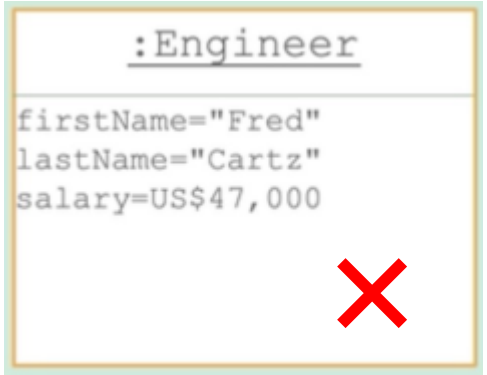
Encapsulation

Il consiste à regrouper les données (attributs) et les méthodes qui agissent sur ces données dans une seule unité, un objet, tout en restreignant l'accès direct aux données. Cela permet de protéger l'état interne d'un objet et de contrôler comment les données sont modifiées.

Protection des données

Contrôle d'accès





Explication :
Les membres de la classe Engineer sont **privée** ➔ on ne peut pas accéder à ces données directement en utilisant le point.

**Code pris depuis la documentation officielle d'Oracle.

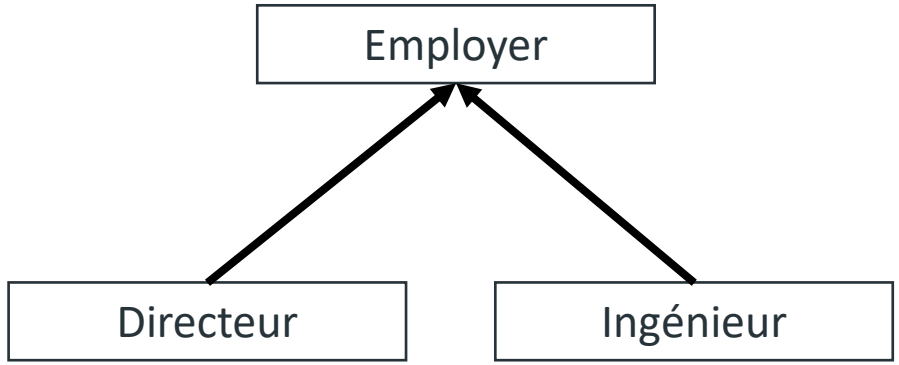
1 - Environnement Java

Introduction

Les quartes principes de la programmation orienté objet sont : abstraction, encapsulation, héritage, polymorphisme

Héritage

L'héritage définit les relations entre les classes dans un langage orienté objet. Un objet d'une sous-classe est un type d'objet de la superclasse.
En Java, toutes les classes héritent de la classe **java.lang.Object** et implémentent ses méthodes tel que toString.



Polymorphisme

- C'est un concept où une variable déclarée avec un type de référence T peut être assignée à différents types d'objets à l'exécution, à condition qu'ils soient des sous-types du type de la variable T.
- Autorise l'utilisation d'une interface ou d'une méthode de plusieurs manières, permettant à différents objets de réagir différemment à une même action,

Chap. I

1 - Environnement Java

C’est quoi Java ?

- C'est un langage de programmation généraliste, similaire à C et C++ ;
- Il est conçu pour être **simple**, **sécurisé** et **portable ;**
- Il est orienté objet et indépendant des plateformes ;
- Actuellement, il est très utilisé dans les applications moderne lié à l’internet des objets, le cloud computing, etc.
- Ce cours couvre Java SE (Standard Edition)

Chap. I

1 - Environnement Java

C’est quoi Java ?

- Il est célèbre pour son principe "écris une fois, exécute partout" (Write Once, Run Anywhere) ➔ un programme Java peut être exécuter sur les différentes plateformes sans modification du code.
- Il offre une gestion automatique de la mémoire grâce à son **ramasse-miettes** (garbage collector), ce qui simplifie la gestion des ressources pour les développeurs.
- Il est largement utilisé dans le développement d'applications web et d'API, grâce à des frameworks populaires comme **Spring** et **Hibernate**.
- Java repose sur l'architecture **Java Virtual Machine (JVM)**, qui permet d'exécuter du code Java indépendamment du système d'exploitation sous-jacent.

1 - Environnement Java

C’est quoi Java ?

les différentes éditions de Java :

- **Java SE (Standard Edition)** : Utilisé pour développer des applications de bureau et serveurs. C'est la base pour tout développement Java ;
- **Java EE (Enterprise Edition)** : Conçu pour les applications d'entreprise à grande échelle, avec des fonctionnalités pour le développement d'applications web, transactions et sécurité ;
- **Java ME (Micro Edition)** : Adapté aux dispositifs à ressources limitées comme les smartphones, appareils embarqués et IoT ;
- **JavaFX** : Destiné au développement d'applications riches en interface graphique (GUI) pour les environnements de bureau et web.
- **Java Card** est utilisé pour les cartes à puce et les appareils de sécurité ;
- **Java MicroProfile** est toujours maintenu et utilisé principalement pour le développement de microservices dans les environnements d'entreprise.

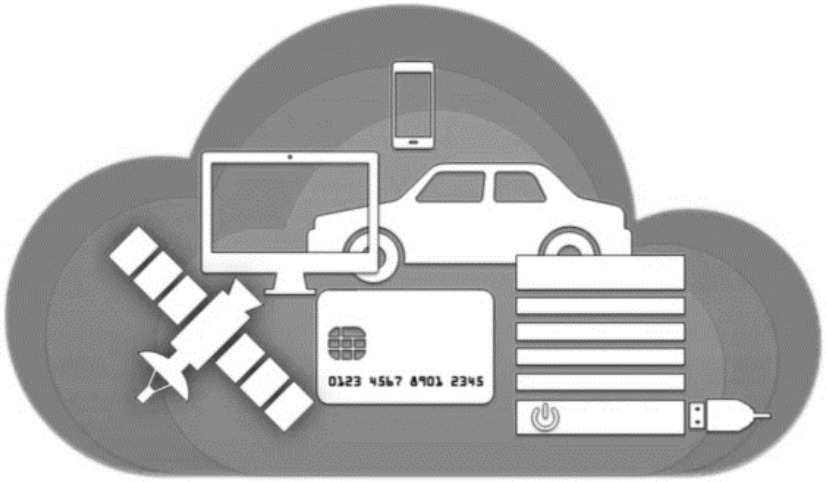


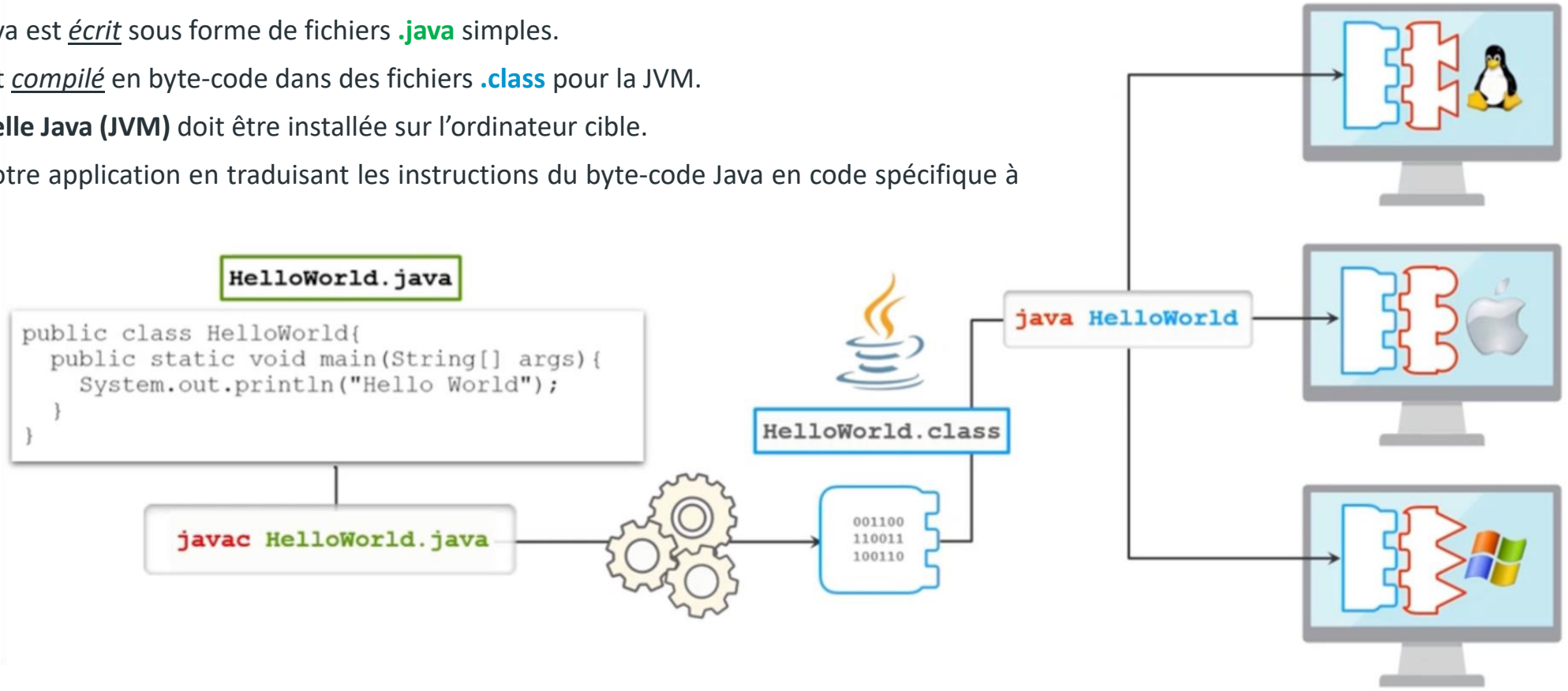
**Image prise depuis la documentation officielle d'Oracle.

1 - Environnement Java

C’est quoi Java ?

Java est un langage de programmation indépendant de la plateforme.

- Le **code source** Java est *écrit* sous forme de fichiers **.java** simples.
- Le **code source** est *compilé* en byte-code dans des fichiers **.class** pour la JVM.
- La **Machine Virtuelle Java (JVM)** doit être installée sur l’ordinateur cible.
- La JVM exécute votre application en traduisant les instructions du byte-code Java en code spécifique à la plateforme.



1 - Environnement Java

C’est quoi Java ?

Pourquoi installer le JDK ?

Le Kit de Développement Java (JDK) contient le logiciel minimum nécessaire pour développer en Java. Les commandes clés incluent :

- **javac** : Convertit les fichiers source .java en bytecode .class
- **java** : Exécute le programme
- **jar** : Regroupe les fichiers
- **javadoc** : Génère la documentation

Vérifier votre version java installer

S’assurer que vous avez la bonne version de Java dans votre path. Executer une des commandes suivantes :

- ✓ *javac - version*
- ✓ *java - version*

1 - Environnement Java

C’est quoi Java ?

Comment installer le JDK ?

1 Télécharger le JDK

- Rendez-vous sur le site officiel d’Oracle pour télécharger la dernière version du JDK [1].
- Choisissez la version correspondant à votre système d’exploitation (Windows, macOS, Linux).

2 Installer le JDK

- **Pour Windows :** Exécutez le fichier d’installation téléchargé et suivez les instructions de l’assistant d’installation.
- **Pour macOS :** Ouvrez le fichier .dmg téléchargé et suivez les instructions pour installer le JDK.
- **Pour Linux :** Décompressez le fichier téléchargé et suivez les instructions spécifiques à votre distribution.

3 Configurer les variables d’environnement

- **Windows :** Ajoutez le chemin du répertoire bin du JDK à la variable d’environnement PATH. Créez également une variable JAVA_HOME pointant vers le répertoire d’installation du JDK.
- **macOS et Linux :** Ajoutez les lignes suivantes à votre fichier de configuration de shell (comme .bashrc ou .zshrc) :

```
export JAVA_HOME=/chemin/vers/jdk export
PATH=$JAVA_HOME/bin:$PATH
```

4 Vérifiez l’installation

- Utiliser une des deux commandes :

java -version **ou** javac -version

1 - Environnement Java

C'est quoi Java ?

Classe Vs Objet ?

- **Les classes** et les objets sont deux concepts clés de la programmation orientée objet.
- Le code Java est structuré avec des **classes**. ➔ un élément de base pour la prog. O. O. avec java
- Une classe représente un type de chose ou un concept, tel qu'un Chien, Chat, Personne, etc.
- Chaque classe définit le type **d'informations (attributs)** qu'elle peut stocker :
  - Un Chien pourrait avoir un nom, une couleur, une taille
  - Une Balle aurait un type, un matériau, etc.
- Chaque classe définit le type de **comportements (opérations)** contenant la logique du **programme (algorithmes)** qu'elle est capable de réaliser :
  - Un Chien pourrait aboyer et rapporter une balle
  - Un Chat pourrait miauler mais il est peu probable qu'il joue à rapporter
- Pour utiliser une classe il faut créer un objet.

```
class Person{

    void play(){
        Dog dog = new Dog();
        dog.name = "Rex";
        Ball ball = new Ball();
        dog.fetch(ball);
    }

}

class Dog{

    String name;

    void fetch(Ball ball){
        ball.find();
        ball.chew();
    }

}
```

1 - Environnement Java

C’est quoi Java ?

Classe Vs Objet ?

- **Un objet** est une **instance** spécifique d’exécution (exemple de) d’une classe en mémoire.
- Tous les objets des différents classes représentent ***l’état du programme***.
- Chaque objet peut avoir des valeurs spécifiques pour chaque attribut défini par une classe qui représente son type. Par exemple :
  - Un chien pourrait s’appeler Fido, être marron et petit
  - Un autre pourrait s’appeler Rex, être orange et grand
- Pour manipuler un objet, vous pouvez le référencer en utilisant une variable du type approprié.
- Une **référence** est une variable qui pointe vers un objet.

1 - Environnement Java

C’est quoi Java ?

Les mots clés réservés

Les mots clés ne peuvent pas être utilisés comme un identifiant (nom des attributs, des méthodes et des classes)

Keywords available since 1.0
Keywords no longer in use
Keywords added in 1.2
Keywords added in 1.4
Keywords added in 5.0
Keywords added in 9.0

Reserved words for literals values

Special identifier added in 10

if	implements	boolean	assert
else	extends	try	enum
continue	interface	catch	module
break	class	finally	requires
for	static	throw	transitive
do	final	throws	exports to
while	return	new	uses
switch	transient	this	provides
case	void	super	with
default	byte	instanceof	opens to
private	short	native	
protected	int	synchronized	true
public	long	volatile	false
import	char	goto	null
package	float	const	
abstract	double	strictfp	var

1 - Environnement Java

C’est quoi Java ?

Convention de nommage en Java

- Java est sensible à la casse ; ***Personne*** n'est pas la même chose que ***personne***
- Le **nom du package** est **l'inverse** du nom de domaine de votre entreprise, plus le système de dénomination adopté au sein de votre entreprise.
- Le **nom de la classe** doit être un nom, en majuscules et minuscules **avec la première lettre** de chaque mot en majuscule
- Le **nom de la variable** doit être un nom, en majuscules et minuscules **avec la première lettre** de chaque mot en majuscule. Il doit être en majuscules et minuscules et commence par une lettre minuscule ;
- Les **noms** **ne doivent** pas commencer par des caractères numériques (0-9), des traits de soulignement ‘ _’ ou des symboles dollar ‘$’ ;
- Le **nom de la constante** est généralement **écrit en majuscules** avec des **traits de soulignement** entre les mots
- Le **nom de la méthode** doit être **un verbe**, en majuscules et minuscules et commencer par une lettre minuscule ; les autres mots commencent par des majuscules,

```
package:   com.oracle.demos.animals
class:     ShepherdDog
variable:  shepherdDog
constant:  MIN_SIZE
method:    giveMePaw
```

```
package:   animals
class:     Shepherd Dog
variable:  _price
constant:  minSize
method:    xyz
```

1 - Environnement Java

C’est quoi Java ?

Règles de syntaxe de base de Java

- Toutes les instructions doivent finir par un point virgule (;).
- les blocs de code doivent être entourés de « { » et de « } ».
- les symboles d'indentation et d'espaces facilitent la lisibilité, mais ne sont pas significatifs syntaxiquement.

```java
public class Main {
    public static void main(String[] args) {
        Animal monChien = new Chien();
        Animal monOiseau = new Oiseau();

        monChien.emettreSon();  // Affiche : Le chien aboie.
        monOiseau.emettreSon(); // Affiche : L'oiseau chante.
    }
}
```

2 – Structure d'une classe

Définition

- Le **nom de la classe** est généralement représenté par un ou plusieurs noms, par exemple, Dog, AnimalCat, Person.
- La classe doit être enregistrée dans un fichier portant le même nom que la classe et l'extension .java.
- Les classes sont regroupées dans des packages.
- Les **packages** sont représentés sous forme de dossiers où les fichiers de classes sont sauvegardés.
- Le nom du package et de la classe doivent former une combinaison ***unique***.

```
package <nom du package>;

class <NomDeLaClasse> {

}
```

Exemple : /somepath/com/oracle/demos/animals/Dog.java

```
package com.oracle.demos.animals;

class Dog {
    // le reste du code de cette classe
}
```

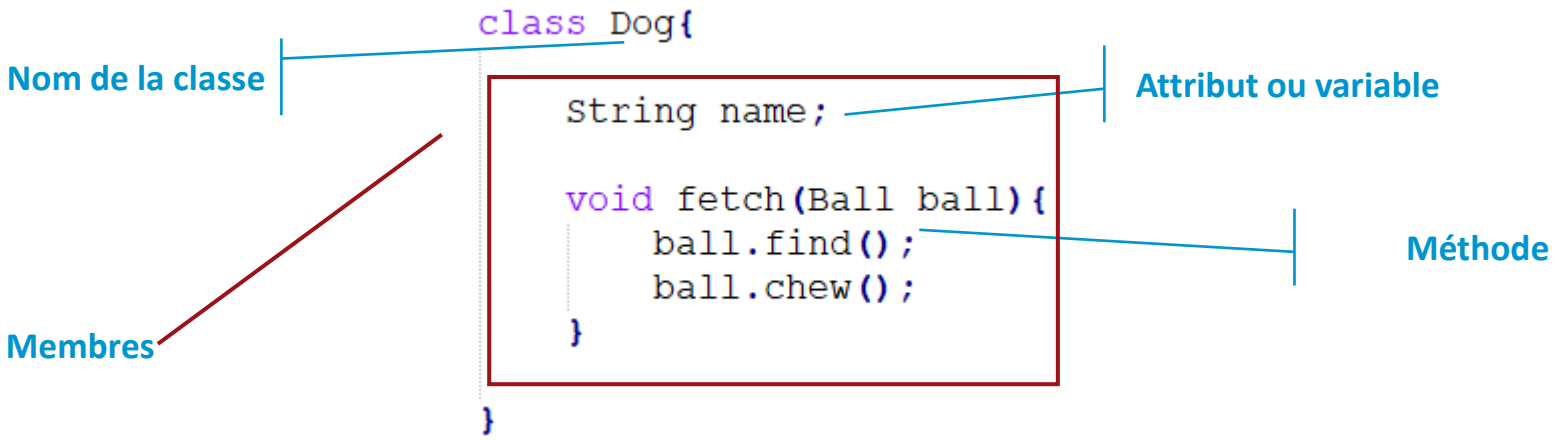
Si la définition du package est manquante
la classe appartiendra à un package "**par défaut**" et ne sera placée dans aucun dossier de package. Cependant, ce n'est pas une pratique recommandée.

2 – Structure d'une classe

Membres de classe

attributs & méthodes

- Les classes Java comportent deux éléments principaux :
  - **les méthodes :** souvent appelées fonctions ou procédures dans d'autres langages ;
  - **les champs :** plus généralement appelés attributs ou variables.
- Ensemble, ils sont appelés les membres de la classe.
- Les attributs contiennent l'état du programme et les méthodes agissent sur cet état.



2 – Structure d'une classe

Membres de classe

attributs & méthodes

Exemple : analyser le code

```
public class Animal {

    String name;

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        name = newName;
    }

}
```

Note :
Le nom de la méthode avec le type des paramètres est appelé : ***la signature de la méthode***

2 – Structure d'une classe

Membres de classe

les commentaires

- Une autre partie courante du code est appelée commentaire.
- Les commentaires ne sont pas du code exécutable, ils peuvent être placés à de nombreux endroits.
- Les commentaires permet de rendre le code plus facile à lire.
- Il existe trois type de commentaire :

Commentaire en une seul ligne

Commentaire en plusieurs ligne

```
// commenter jusqu'à la fin de la ligne

/* commentaire
 * sur plusieurs lignes
 *
 */

/**
 * Javadoc multiple- line comment
 * @author Jeanne and Scott
 */
```

Commentaire en plusieurs ligne

- Il est similaire à un commentaire multiligne, sauf qu'il commence par **/****.
- Cette syntaxe spéciale indique à l'outil Javadoc de prêter attention au commentaire.
- Les commentaires Javadoc ont une structure spécifique que l'outil Javadoc sait lire.

2 – Structure d'une classe

Les modificateurs d'accès

- Les **modificateurs d'accès** décrivent **la visibilité** des classes, des variables et des méthodes.
  - **public :** Visible pour toute autre classe
  - **protected :** Visible pour les classes du même package ou pour les sous-classes
  - **<default>** : Visible uniquement pour les classes dans le même package
  - **private :** Visible uniquement au sein de la même classe
- **Notes :** <default> signifie qu'aucun modificateur d'accès n'est explicitement défini.

Manière d'utilisation

```
package <package name>;
import <package name>.<class name>;
import <package name>.*;
<access modifier> class <ClassName> {
    <access modifier> <variable definition>
    <access modifier> <method definition>
}
```

Exemple

```
package b;
public class Y extends X {
    public void doThings() {
        Y p = new Y();
        p.m1="Hello";
        p.m2="Hello";
        p.m3="Hello";
        p.m4="Hello";
    }
}
```

```
package a;
public class X {
    public String m1;
    protected String m2;
    String m3;
    private String m4;
}
```

2 – Structure d'une classe

Accès à une classe

- **Préfixer le nom de la classe avec le nom du package.**
- Utiliser l'instruction **import** pour importer **des classes spécifiques** ou **tout le contenu du package**.
  - L'importation de toutes les classes du package java.lang.* est implicitement assumée.
- L'exemple montre trois façons alternatives de référencer la classe Dog dans le package animals à partir de la classe Owner dans le package people :

```
package animals;
public class Dog {
}
```

```
package people;
public class Owner {
    animals.Dog myDog;
}
```

```
package people;
import animals.Dog;
public class Owner {
    Dog myDog;
}
```

```
package people;
import animals.*;
public class Owner {
    Dog myDog;
}
```

2 – Structure d'une classe

Classe Main

- La **classe Main** est une classe doté de la méthode main.
- La **méthode principale (main)** est le point d'entrée (de départ de l'exécution) dans une application java.
- Le nom de la méthode doit être **main**.
- Elle doit être :
  - **Public :** vous avez l'intention d'invoquer cette méthode depuis l'extérieur de cette classe.
  - **Static :** Ces méthodes peuvent être invoquées sans créer une instance de cette classe.
  - **Void : e**lle ne retourne pas de valeur.
- Elle doit accepter **un tableau d'objets String** comme **seul paramètre**. (Le nom de ce paramètre, args, est sans importance.)

```
package demos;

public class Whatever {
    public static void main(String[] args) {
        // L'exécution du programme commence ici
    }
}
```

Toutes les déclarations suivantes sont possibles :
String[] args
String options[]
String... Friends ➔les « … » sont appelés varargs

2 – Structure d'une classe

Classe Main

Passage des paramètres au main

Soit l'exemple suivant :

```
package demos;

public class Whatever {
    public static void main(String[] args) {
        System.out.println(args[0]);
        System.out.println(args[1]);
    }
}
```

Accès au premier paramètre du tableau args

- Pour l'exécuter, tapez ceci :
  javac Whatever.java
  java Whatever Bronx Zoo
- Le résultat est ce à quoi vous pourriez vous attendre :
  Bronx
  Zoo
- Le programme identifie correctement les deux premiers « mots » comme arguments.
- Des espaces sont utilisés pour séparer les arguments.
- Pour avoir des espaces à l'intérieur d'un argument, il faut utiliser des guillemets comme dans cet exemple :
  javac Zoo.java
  java Zoo "San Diego" Zoo
- Maintenant il y a un espace dans le résultat :
  San Diego
  Zoo

2 – Structure d'une classe

Classe Main

Compilation

Les classes sont compilées avec le compilateur javac de Java.

- Le paramètre **-classpath** ou **-cp** pointe vers **l'emplacement** des autres classes qui peuvent être nécessaires pour compiler le code.
- Le paramètre **-d** pointe vers **le chemin de stockage** du résultat de la compilation. (Le compilateur crée des sous-dossiers de packages avec des fichiers de classe compilés dans ce chemin.)
- La dernière donnée est le **chemin vers le fichier source**.
- Exemple :

```
Javac -cp /projet/classes -d /projet/classes / projet/sources/demos/Whatever.java
```

- Une fois compilé, le fichier compilé sera situé :

```
/projet/classes/demos/Whatever.class
```

2 – Structure d'une classe

Classe Main

Compilation & documentation

Commentaires de documentation :

- Peuvent contenir des balises HTML.
- Peuvent contenir des balises descriptives préfixées par @.
- Sont utilisées par l'outil javadoc pour générer la documentation.

```
// single-line comment

/*
   multi-line comment
*/

/**
*  The {@code Whatever} class
*  represents an example of
*  documentation comment
* @version 1.0
* @author oracle
*/
```

Package demos

Class Whatever

java.lang.Object
    demos.Whatever

public class **Whatever**
extends Object

The Whatever class represents an example of documentation comment

Version:

1.0

Author:

oracle

```
javadoc -d <documentation path>
        -sourcepath <source code path>
        -subpackages <name of the root package>
```

2 – Structure d'une classe

Ordre des éléments

L'élément	Exemple	Obligatoire ?	Où il est inséré ?
Déclaration du package	package abc;	Non	Première ligne dans le fichier (on exclut les commentaires ou les lignes vides)
Instruction d'importation	import java.util.*;	Non	Directement après l'instruction de déclaration de package
Déclaration de type de niveau	public class C	Oui	Directement après l'instruction de « import »
Déclaration des attributs	int value;	Non	Tout élément de niveau supérieur dans une classe
Déclaration des méthodes	void method()	Non	Tout élément de niveau supérieur dans une classe

3 – Packages

Introduction

C’est quoi?

Un package est un **mécanisme** pour **organiser** les classes et les interfaces de **manière hiérarchique**.

Il permet d'éviter les conflits de noms en regroupant les classes similaires.

Pourquoi l’utiliser?

- **Modularité :** Facilite l'organisation et la gestion du code source.
- **Réutilisation :** Encourage la réutilisation de classes communes dans différents projets.
- **Encapsulation :** Les classes d'un package peuvent avoir des niveaux d'accès différents, comme private, protected, etc.

Types de packages

- **Packages intégrés :** Par exemple, java.lang, java.util, etc.
- **Packages personnalisés :** Vous pouvez créer vos propres packages

3 – Packages

Déclaration & importation

Déclaration d'un package :
La première ligne de chaque fichier source doit spécifier le package.

```
package com.oracle.demos.animals;

class Dog {
    // le reste du code de cette classe
}
```

Importer des classes depuis un package :
Pour utiliser une classe qui appartient à un autre package, il faut l'importer.

```
//Exemple d'importation d'une classe spécifique
import java.util.ArrayList;

//Pour importer toutes les classes d'un package
import java.util.*;
```

L'importation du package java.lang :

- Implicitement importé dans chaque programme Java (pas besoin de l'importer manuellement).
- Contient des classes essentielles comme String, Math, System.

3 – Packages

Meilleurs pratiques

Convention de nommage des packages :
Utilisez des noms de packages en minuscules et suivez un schéma hiérarchique.
Par exemple : com.exemple.application.

Accessibilité des classes dans un package :
Par défaut, les classes dans un package ne sont accessibles qu'aux autres classes du même package (modificateur d'accès package-private).

```java
// Dans un fichier appelé com/exemple/Bonjour.java
package com.exemple;

public class Bonjour {
    public void saluer() {
        System.out.println("Bonjour le monde !");
    }
}

// Dans un autre fichier
import com.exemple.Bonjour;

public class Test {
    public static void main(String[] args) {
        Bonjour bonjour = new Bonjour();
        bonjour.saluer();
    }
}
```

3 – Packages

Meilleurs pratiques

Exemple :

```
// PAS BON - un caractère générique ne correspond qu'à
// les noms de classe, pas à "file.Files"
import java.nio.*;

// PAS BON - vous ne pouvez avoir qu'un seul caractère générique
// et il doit être à la fin
import java.nio.*.*;

// PAS BON - vous ne pouvez pas importer de méthodes
// uniquement des noms de classe
import java.nio.file.Paths.*;

//utiliser un caractère générique pour importer les deux en même temps.
import java.nio.file.*;

// importer les deux classes explicitement.
import java.nio.file.Files;
import java.nio.file.Paths;

public class InputImports {
    public void read(Files files) {
        Paths.get("name");
    }
}
```

4 – Création des objets

Introduction

Les objets sont des instances de classes en Java.
En Java, un programme manipule des objets créés à partir de classes. Ces objets peuvent interagir entre eux pour réaliser des tâches.

Exemple : création d'un objet de type Animal

```
Animal a = new Animal();
```

Utilisez le mot-clé **new** suivi du nom de la classe et de parenthèses ➔ appel d'un **constructeur**

Cela crée une instance de la classe Animal et la référence est stockée dans la variable a.

4 – Création des objets

Constructeurs

- Ce sont des Méthodes spéciales utilisées pour initialiser un objet.
- Il n'a pas de type de retour et son nom correspond à celui de la classe.
- Il y a deux types de constructeurs :
  - **Par défaut :** Si aucun constructeur n'est défini, Java fournit un constructeur par défaut qui ne fait rien. Le constructeur par défaut est un constructeur sans paramètres.
  - **Personnalisé :** Si votre classe nécessite une logique particulière lors de la création d'objets, vous devrez définir explicitement un constructeur.
    Exemple de situation : Initialisation complexe ou utilisation d'arguments.

```
public class Animal {

    public Animal() {
        System.out.println("dans le constructeur");
    }

}
```

Le constructeur permet d'exécuter du code lors de la création de l'objet, comme afficher un message ici.

Différence entre un constructeur et une méthode :

- Le constructeur n'a pas de type de retour
- public void Animal(){..}➔ce n'est pas un constructeur

4 – Création des objets

Lectures & écritures

Accès direct aux variables d'instance : Les champs d'une classe peuvent être lus et modifiés directement.

```java
public class Swan {

    int numberEggs;

    public static void main(String[] args) {
        Swan mother = new Swan();
        mother.numberEggs = 1;  // Modification de la variable
        System.out.println(mother.numberEggs);  // Lecture de la variable
    }

}
```

Ici, la variable numberEggs est modifiée et lue directement dans la méthode main().

4 – Création des objets

Blocs d'initialisation d'instance

Un bloc d'initialisation d'instance : Ce sont des blocs de code situés en dehors des méthodes, utilisés pour initialiser des variables d'instance.

```
public class Bird {
    { System.out.println("Snowy"); } // Bloc d'initialisation d'instance

    public static void main(String[] args) {
        { System.out.println("Feathers"); }  // Bloc dans méthode
    }

}
```

4 – Création des objets

Ordre d'initialisation

Il faut se rappeler que :

- les attributs et les blocs d'initialisation d'instance sont exécutés dans l'ordre dans lequel ils apparaissent dans le fichier.
- Le constructeur s'exécute une fois que tous les champs et les blocs d'initialisation d'instance ont été exécutés.

```
public class Chick {

    private String name = "Fluffy";

    { System.out.println("setting field"); }

    public Chick() {
        name = "Tiny";
        System.out.println("setting constructor");
    }

    public static void main(String[] args) {
        Chick chick = new Chick();
        System.out.println(chick.name);
    }

}
```

Résultat de l'exécution

```
setting field
setting constructor
Tiny
```

4 – Création des objets

Ordre d'initialisation

Exemple : donner le résultat de l'exécution de ce code

```
public class Egg {

    public Egg() {
        number = 5;
    }

    public static void main(String[] args) {
        Egg egg = new Egg();
        System.out.println(egg.number);
    }

    private int number = 3;
    { number = 4; }

}
```

5 – Types de données

Types primitifs

- Il existe deux types de données en java : types **primitifs**, types de **référence**
- Java possède **huit types** de données intégrés, appelés types primitifs.
- Se sont les éléments de base pour la création des objets en Java.
- Tous les objets Java sont des collections complexes de ces types primitifs.
- Un type primitif n'est pas un objet en Java.
- Un primitif représente une valeur simple en mémoire, comme un nombre ou un caractère.

5 – Types de données

Types primitifs

Keyword	Type	Min value	Max value	Default value	Example
boolean	true or false	n/a	n/a	false	true
byte	8-bit integral value	-128	127	0	123
short	16-bit integral value	-32,768	32,767	0	123
int	32-bit integral value	-2,147,483,648	2,147,483,647	0	123
long	64-bit integral value	-2^{63}	$2^{63} - 1$	0L	123L
float	32-bit floating-point value	n/a	n/a	0.0f	123.45f
double	64-bit floating-point value	n/a	n/a	0.0	123.456
char	16-bit Unicode value	0	65,535	\u0000	'a'

String est-il une primitive ?

Non, ce n'est pas le cas. Cela dit, String est souvent confondu avec une neuvième primitive, car Java inclut une prise en charge intégrée des littéraux et des opérateurs de chaîne.

À retenir : il s'agit d'un objet et non d'une primitive.

5 – Types de données

Types primitifs

- Un float nécessite la lettre f ou F après le nombre pour que Java sache qu'il s'agit d'un float. Sans f ou F, Java interprète une valeur décimale comme un double.

- Un long nécessite la lettre l ou L après le nombre pour que Java sache qu'il s'agit d'un long. Sans l ou L, Java interprète un nombre sans point décimal comme un int dans la plupart des scénarios.

Nombre et Underscore (_)?

Il est possible de séparer les parties d'un nombre par des '_'. Cependant il ne faut les insérer au début, ni avant et après la virgule (pour les float et double)

```
int million1 = 1000000;
int million2 = 1_000_000;
```

5 – Types de données

Types primitifs

En java, il est possible de spécifier la valeur d'un nombre en utilisant les formats suivants :

- Octal (chiffres de 0 à 7), qui utilise le nombre 0 comme préfixe, par exemple 017.
- Hexadécimal (chiffres de 0 à 9 et lettres A à F/a à f), qui utilise 0x ou 0X comme préfixe, par exemple 0xFF, 0xff, 0XFf. L'hexadécimal ne fait pas la distinction entre les majuscules et les minuscules, donc tous ces exemples signifient la même valeur.
- Binaire (chiffres de 0 à 1), qui utilise le nombre 0 suivi de b ou B comme préfixe, par exemple 0b10, 0B10.

5 – Types de données

Types primitifs

Déclaration

Syntaxe : <type> <nom du vraiable> = <value>

- Une variable peut être déclaré sans initialisation
- Il faut initialiser une variable avant de l'utiliser
- Les valeurs numérique peuvent être exprimé en binaire, en octet, en décimal ou hexadécimal
- valeur double et réel peuvent être exprimé en notation normal ou exponentiel
- l'affectation d'une variable à une autre crée une copie de sa valeur
- les valeur de type char doivent être mentionné entre un simple quotte.

```
int a = 0b101010; // binary
short b = 052;    // octal
byte c = 42;      // decimal
long d = 0x2A;    // hex
float e = 1.99E2F;
double f = 1.99;
long g = 5, h = c;
float i = g;
char j = 'A';
char k = '\u0041', l = '\101';
int s;
s = 77;
```

```
byte a;
byte b = a;
byte c = 128;
int d = 42L;
float e = 1.2;
char f = "a";
char g = 'AB';
boolean h = "true";
boolean i = 'false';
boolean j = 0;
boolean k = False;
```

5 – Types de données

Types de référence

- Un type de référence fait référence à un objet (une instance d'une classe).
- Contrairement aux types primitifs qui stockent leurs valeurs dans la mémoire où la variable est allouée, les références ne stockent pas la valeur de l'objet auquel elles se réfèrent.
- Une référence "pointe" vers un objet en stockant l'adresse mémoire où se trouve l'objet, un concept appelé "pointeur".
- Contrairement à d'autres langages, Java ne permet pas de connaître l'adresse mémoire physique.
- Vous pouvez seulement utiliser la référence pour vous référer à l'objet.
- Ils peuvent être utilisés pour appeler des méthodes, en supposant que la référence n'est pas nulle

5 – Types de données

Types de référence

Exemple :

String greeting;

- La variable **greeting** est une référence qui ne peut pointer que vers un objet de type String. Une valeur peut être assignée à une référence de deux façons :
  - Une référence peut être assignée à un autre objet du même type ou d'un type compatible.
  - Une référence peut être assignée à un nouvel objet en utilisant le mot-clé new.

- Par exemple, l'instruction suivante assigne cette référence à un nouvel objet :

```
greeting = new String("How are you?");
```

- La référence **greeting** pointe vers un nouvel objet String, "How are you?". Cet objet String n'a pas de nom et ne peut être accessible qu'à travers une référence correspondante.

5 – Types de données

Types de référence

Classes enveloppe :

Primitive type	Wrapper class	Wrapper class inherits Number?	Example of creating
boolean	Boolean	No	Boolean.valueOf(true)
byte	Byte	Yes	Byte.valueOf((byte) 1)
short	Short	Yes	Short.valueOf((short) 1)
int	Integer	Yes	Integer.valueOf(1)
long	Long	Yes	Long.valueOf(1)
float	Float	Yes	Float.valueOf((float) 1.0)
double	Double	Yes	Double.valueOf(1.0)
char	Character	No	Character.valueOf('c')

5 – Types de données

Type avec var

Vous avez la possibilité d'utiliser le mot-clé **var** au lieu du type lors de la déclaration de variables locales dans certaines conditions.

```
public class Zoo {
    public void whatTypeAmI() {
        var name = "Hello";
        var size = 7;
    }
}
```

- Le nom formel de cette fonctionnalité est l'inférence de type pour les variables locales.
- la déclaration et l'initialisation de silly doivent se produire sur la même ligne.
- Vous ne pouvez utiliser cette fonctionnalité que pour les variables locales.

```
public void reassignment() {
    var number = 7;
    number = 4;
    number = "five";  // DOES NOT COMPILE
}
```

```
public class VarKeyword {
    var tricky = "Hello"; // NE COMPILE PAS
}
```

5 – Types de données

String & Text

- La classe **java.lang.String** représente une séquence de caractères.
- Ses instances représentent des séquences de caractères.
- Comme tout autre objet Java, l'objet String peut être **instancié** en utilisant le mot-clé **new**.
- Cependant, String est le seul objet Java qui permet une instanciation simplifiée sous forme de valeur de texte entourée de guillemets doubles : "some text" et c'est l'approche recommandée.

5 – Types de données

String & Text

- Les objets **String** sont immuables.
- Une fois qu'un objet String est initialisé, il ne peut pas être modifié.
- Les opérations sur les chaînes de caractères telles que trim(), concat(), toLowerCase(), toUpperCase(), et ainsi de suite, renverront toujours un nouvel objet String, mais ne modifieront pas la chaîne d'origine.
- Il est possible de réassigner la référence de la chaîne pour pointer vers un autre objet String.
- Pour des raisons de commodité, String permet l'utilisation de l'opérateur + au lieu de la méthode concat(). Cependant, rappelez-vous que + est aussi un opérateur arithmétique.

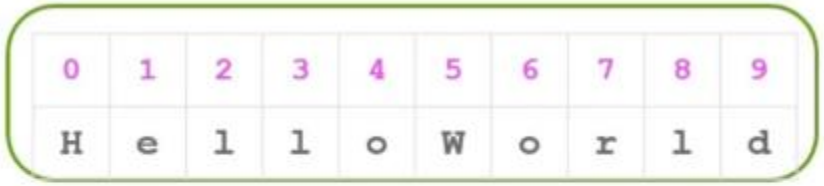

```
String s = "";
s=1+1+"u";    // s is 2u
s="u"+1+1;    // s is u11
s="u"+(1+1); // s is u2
```

5 – Types de données

String & Text

Les chaînes contiennent une séquence de caractères indexée par un entier.

- L'index d'une chaîne commence à 0.
- Lorsque vous obtenez une sous-chaîne, l'index de début est inclus dans le résultat, mais l'index de fin ne l'est pas.
- Si une sous-chaîne n'est pas trouvée, la méthode indexOf renvoie -1.
- Les opérations indexOf et lastIndexOf sont surchargées (ont plus d'une version) et acceptent un caractère ou un paramètre String, et peuvent aussi accepter une recherche à partir de la position de l'index.
- Une tentative d'accès à un texte au-delà de la dernière position d'index valide (longueur - 1) entraînera une exception.




```
String a = "HelloWorld";
String b = a.substring(0,5); // b is "Hello"
int c = a.indexOf('o');      // c is 4
int d = a.indexOf('o',5);    // d is 6
int e = a.lastIndexOf('l');  // e is 8
int f = a.indexOf('a');      // f is -1
char g = a.charAt(0);        // g is H
int h = a.length();          // h is 10
char i = a.charAt(10);

      throw StringIndexOutOfBoundsException
```

5 – Types de données

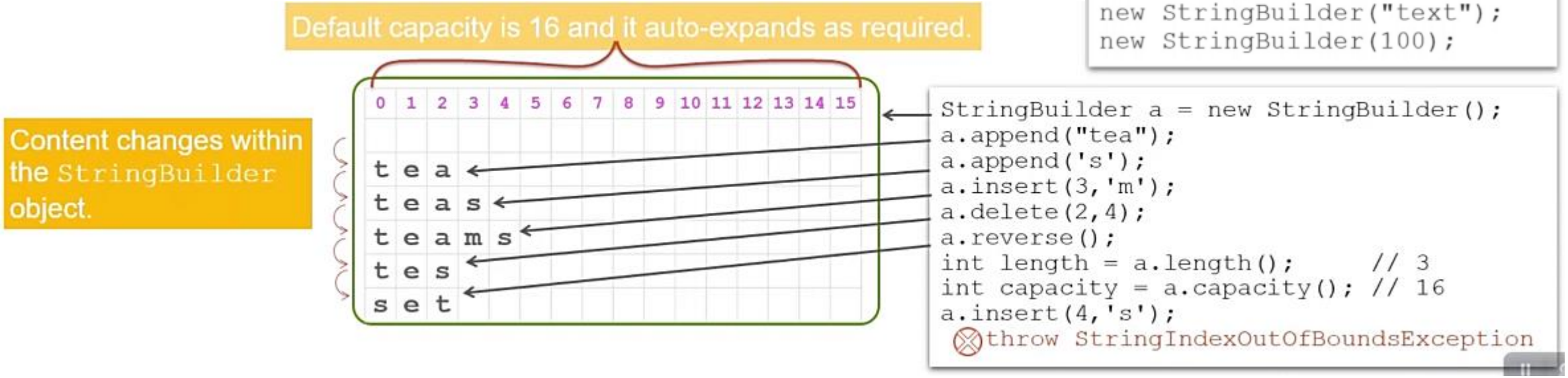
String & Text

StringBuilder

Une autre façon de manipuler du texte en Java est avec la classe java.lang.StringBuilder.

- Les objets StringBuilder sont mutables, ce qui permet de modifier les séquences de caractères qu'ils stockent.
- Manipuler des modifications de texte avec StringBuilder réduit le nombre d'objets String que vous devez créer.
- Certaines méthodes comme substring, indexOf, charAt sont identiques à celles de la classe String.
- Des méthodes supplémentaires sont disponibles : append, insert, delete, reverse, acceptant des paramètres String ou char.
- La séquence de caractères doit être continue. Elle peut contenir des espaces, mais vous ne pouvez pas laisser d'intervalles sans caractères.
- Comme pour d'autres classes, les objets StringBuilder sont instanciés à l'aide du mot-clé new.

```
new StringBuilder();
new StringBuilder("text");
new StringBuilder(100);
```



5 – Types de données

String & Text

StringBuilder

Un bloc de texte est une chaîne littérale multiligne qui évite le recours à la plupart des séquences d'échappement => formate automatiquement la chaîne de manière prévisible et donne au développeur le contrôle du format lorsqu'il le souhaite.

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello, world</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

```
String html = """
              <html>
                  <body>
                      <p>Hello, world</p>
                  </body>
              </html>
              """;
```

5 – Types de données

Local date & time

L'API de date et d'heure locale est utilisée pour gérer les valeurs de date et d'heure :

- Classes **LocalDate**, **LocalTime** et **LocalDateTime** du package java.time
- Les objets de date et d'heure peuvent être créés à l'aide des méthodes :
  - **now()** pour obtenir la date et l'heure actuelle ,
  - **LocalDateTime.of**(year, month, day, hours, minutes, seconds, nanoseconds)
  - **LocalTime.of**(hours, minutes, seconds, nanoseconds)
  - **LocalDate.of**(year, month, day) pour une date et une heure spécifiques
  - en combinant d'autres objets de date et d'heure à l'aide de **atTime()** et **of(localDate, localTime)** ou en extrayant des parties de date et d'heure de LocalDateTime à l'aide de **toLocalDate()** et **toLocalTime()**

```
LocalDate today = LocalDate.now();
LocalTime thisTime = LocalTime.now();
LocalDateTime currentDateTime = LocalDateTime.now();
LocalDate someDay = LocalDate.of(2019,Month.APRIL,1);
LocalTime someTime = LocalTime.of(10,6);
LocalDateTime otherDateTime = LocalDateTime.of(2019,Month.MARCH,31,23,59);
LocalDateTime someDateTime = someDay.atTime(someTime);
LocalDate whatWasTheDate = someDateTime.toLocalDate();
```

5 – Types de données

Local date & time

Caractéristiques des opérations de date et d'heure locales

- Les objets Date et Heure locaux sont immuables
- De nouveaux objets de date et d'heure peuvent être produits à partir d'objets existants à l'aide des méthodes **plusXXX**(), **minusXXX()** ou **withXXX()**
- Il est possible d'enchaîner les invocations de méthodes, car toutes les opérations de manipulation de date et d'heure renvoient des objets de date et d'heure
- Des portions individuelles d'objets de date et d'heure peuvent être récupérées avec les méthodes **getXXX().**
- Les opérations **isBefore()** et **isAfter()** vérifient si une date ou une heure est antérieure ou postérieure à une autre.

```
LocalDateTime current = LocalDateTime.now();
LocalDateTime different =  current.withMinute(14).withDayOfMonth(3).plusHours(12);
int year = current.getYear();
boolean before = current.isBefore(different);
```

5 – Types de données

Local date & time

Instants, Durées et Périodes

l'API peut également représenter des périodes et des durées de temps.

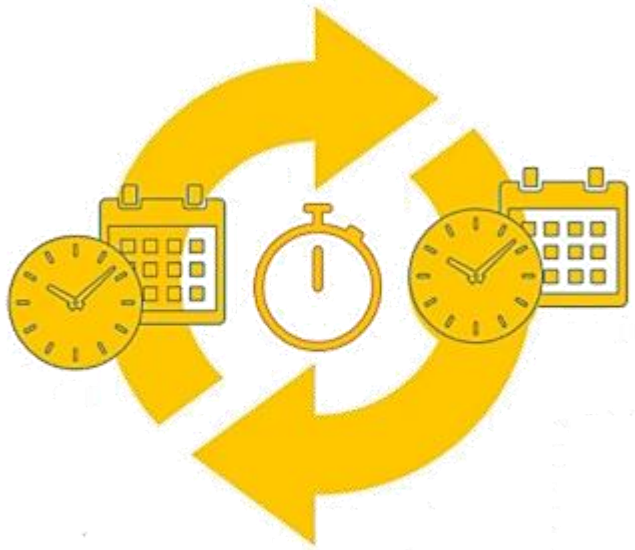
- La classe **java.time.Duration** peut représenter une quantité de temps en nanosecondes.
- La classe **java.time.Period** peut représenter une quantité de temps en unités telles que les années ou les jours.
- La classe **java.time.Instant** peut représenter un point instantané sur la ligne de temps (un horodatage).
- Les objets Duration, Period et Instant sont **immuables**.
- Elles fournissent des méthodes telles que now(), ofXXX(), plusXXX(), minusXXX(), withXXX() et getXXX().
- Elles fournissent des méthodes telles que between() et isNegative() pour gérer les distances entre des points dans le temps.
- Utilisez des techniques de codage identiques telles que le chaînage de méthodes.

5 – Types de données

Local date & time

Instants, Durées et Périodes

exemple



```java
// Date actuelle
LocalDate aujourdHui = LocalDate.now();

// 1er avril 2019
LocalDate jourDuPoissonAvril = LocalDate.of(2019, Month.APRIL, 1);

// Instant actuel
Instant horodatage = Instant.now();

// Nanosecondes depuis la dernière seconde
int nanoSecondesDepuisLaDerniereSeconde = horodatage.getNano();

// Période entre le 1er avril 2019 et aujourd'hui
Period combienDeTemps = Period.between(jourDuPoissonAvril, aujourdHui);

// Durée de 2 heures
Duration deuxHeures = Duration.ofHours(2);

// Durée en secondes après avoir soustrait 15 minutes
long secondes = deuxHeures.minusMinutes(15).getSeconds();

// Nombre de jours dans la période
int jours = combienDeTemps.getDays();
```

5 – Types de données

Destruction des objets

Java propose un garbage collector pour rechercher automatiquement les objets qui ne sont plus nécessaires.
Tous les objets Java sont stockés dans le *heap* de la mémoire d'un programme.
Le *heap*, également appelé *free store*, représente un grand réservoir de mémoire inutilisée allouée à votre application Java.
Si votre programme continue à instancier des objets et à les laisser sur le heap, il finira par manquer de mémoire et planter ➔ intervention du ***grabage collector*** pour vider l'espace inutilisé

6 – Les opérateurs

Liste des opérateurs Java par ordre de priorité.

Operators	Precedence
postfix increment and decrement	++ --
prefix increment and decrement, and unary	++ -- + - ~ !
multiplicative	* / %
additive	+ -
bit shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

6 – Les opérateurs

Liste des opérateurs Java par ordre de priorité.

```
int a = 2, b = 3;
int c = b-a*b;   // (c is -3)
int d = (b-a)*b; // (d is 3)
```

```
int a = 1, b = 0;
a++;      // increment (a is 2)
++a;      // increment (a is 3)
a--;      // decrement (a is 2)
--a;      // decrement (a is 1)
b = a++; // increment postfix (b is 1, a is 2)
b = ++a; // increment prefix  (b is 3, a is 3)
b = a--; // increment postfix (b is 3, a is 2)
b = --a; // increment prefix  (b is 1, a is 1)
```

```
int a = 1;   // assignment (a is 1)
int b = a+4; // addition (b is 5)
int c = b-2; // subtraction (c is 3)
int d = c*b; // multiplication (d is 15)
int e = d/c; // division (e is 5)
int f = d%6; // modulus (f is 3)
```

```
int a = 1, b = 3;
a += b; // equivalent of a=a+b (a is 4)
a -= 2; // equivalent of a=a-2 (a is 2)
a *= b; // equivalent of a=a*b (a is 6)
a /= 2; // equivalent of a=a/2 (a is 3)
a %= a; // equivalent of a=a%a (a is 0)
```

6 – Les opérateurs

Exemple :

```
byte a = 127, b = 5;
byte c = a+b;              // compilation fails
          int
int d = a + b;             // d is 132
byte e = (byte)(a+b);      // e is -124  (type overflow, because 127 is the max byte value)
int f = a/b;               // f is 25    (a/b is 25 because it is an int)
float g = a/b;             // g is 25.0F (result of the a/b can be implicitly or
float h = (float)(a/b);    // h is 25.0F  explicitly casted to float, but a/b is still 25)
float i = (float)a/b;      // i is 25.4F (when either a or b
float j = a/(float)b;      // j is 25.4F  is float the a/b becomes float)
b = (byte)(b+1);           // explicit casting is required, because b+1 is an int
b++;                       // no casting is required for ++ and -- operators
char x = 'x';
char y = ++x;              // arithmetic operations work with character codes
```

6 – Les opérateurs

La classe **java.lang.Math** fournit

diverses opérations mathématiques

Examples of Math functions:

```
double a = 1.99, b = 2.99, c = 0;
c = Math.cos(a);    // cosine
c = Math.acos(a);   // arc cosine
c = Math.sin(a);    // sine
c = Math.asin(a);   // arc sine
c = Math.tan(a);    // tangent
c = Math.atan(a);   // arc tangent
c = Math.exp(a);    // e^a
c = Math.max(a,b);  // greater of two values
c = Math.min(a,b);  // smaller of two values
c = Math.pow(a,b);  // a^b
c = Math.sqrt(a);   // square root
c = Math.random();  // random number 0.0>=c<1.0
```

Numeric rounding example :

```
int a = 11, b = 3;
long c = Math.round(a/b);                    // c is 3
double d = Math.round(a/b);                  // d is 3.0
double e = Math.round((double)a/b*100)/100.0; // e is 3.67
```

6 – Les opérateurs

If … else

- Le bloc de if est exécuté lorsque l'expression booléenne renvoie true sinon le bloc else est exécuté
- la clause else est facultative
- il n'y a pas d'opérateur else/if en Java, mais vous pouvez intégrer un if/else dans une autre construction if/else

```
if (<boolean expression>) {
  /* logic to be executed when
     if expression yields true */
} else {
  /* logic to be executed when
     if expression yields false */
}
```

```
int a = 2, b = 3;
if (a > b) {          // is false
  a--;                // not executed
} else {              // algorithm enters else block
  if (a < b) {        // is true
    a++;              // a is 3
  } else {            // this else block is not executed
    b++;              // not executed
  }
}
```

```
int a = 2, b = 3;
if (a > b)
  a--;
  b++;
else
  a++;
```

Quand le block de if ou de else contient une seule instruction, l'utilisation de {} devient optionnel

```
int a = 2, b = 3;
if (a > b)
  a--;
else if (a < b)
  a++;
else
  b++;
```

6 – Les opérateurs

If … else

- **L'opérateur ternaire** est utilisé pour effectuer une affectation conditionnelle. si vous avez seulement besoin d'attribuer une valeur basée sur une condition,
- Lorsque l'expression booléenne donne true, la valeur après **?** est attribuée. Lorsque l'expression booléenne donne false, la valeur après **:** est attribuée

```
<variable> = (<boolean expression>) ? <value one> : <value two>;
```

```
int a = 2, b = 3;
int c = (a >= b) ? a : b; // c is 3
```

Les deux codes donnent le même résultat

```
int a = 2, b = 3;
int c = 0;
if (a >= b) {
  c = a;
} else {
  c = b;
}
```

6 – Les opérateurs

If … else

- **L'opérateur ternaire** est utilisé pour effectuer une affectation conditionnelle. si vous avez seulement besoin d'attribuer une valeur basée sur une condition,
- Lorsque l'expression booléenne donne true, la valeur après **?** est attribuée. Lorsque l'expression booléenne donne false, la valeur après **:** est attribuée

```
<variable> = (<boolean expression>) ? <value one> : <value two>;
```

```
int a = 2, b = 3;
int c = (a >= b) ? a : b; // c is 3
```

Les deux codes donnent le même résultat

```
int a = 2, b = 3;
int c = 0;
if (a >= b) {
  c = a;
} else {
  c = b;
}
```

6 – Les opérateurs

Switch

- L'expression de **switch** doit être de l'un des types suivants : byte, short, int, char, String, enum
- Les choix doivent correspondre au type d'expression.
- Le flux d'exécution passe au cas dans lequel le choix correspond à la valeur de l'expression.
- Le flux d'exécution continue jusqu'à ce qu'il atteigne la fin de switch ou rencontre une instruction break facultative.
- Si l'expression de switch ne correspond à aucun des cas, le cas par défaut est exécuté. Il n'est pas nécessaire qu'il s'agisse du dernier cas de la séquence et il est facultatif.

```
switch (<expression>)
{
  case <label>:
    <case logic>
  case <label>:
    <case logic>
    break;
  default:
    <case logic>
}
```

```
char status = 'N';
double price = 10;
switch (status) {
  case 'S':
    price += 1; // not executed
  case 'N':
    price += 2; // price is 12
  case 'D':
    price -= 4; // price is 8
    break;
  case 'E':
    price = 0;  // not executed
    break;
  default:
    price = 3;  // not executed
}
//the rest of the program logic
```

6 – Les opérateurs

Switch

Comme toutes les expressions, les expressions switch évaluent une valeur unique et peuvent être utilisées dans des instructions.

- Peut contenir des étiquettes « case L -> »
- élimine le besoin d'instructions break
- Un case peut lister plusieurs valeurs séparées par des virgules.
- Un vrai case peut exécuter une ligne ou un bloc de code.

```
switch(month) {
   case 1, 3, 5, 7, 8, 10, 12 -> System.out.print("31 days.");
   case 4, 6, 9, 11           -> System.out.print("30 days.");
   case 2 -> {
       if (!isLeapYear) {
          System.out.print("28 days.");
       } else
          System.out.print("29 days.");
   }
   default ->  System.out.print("Invalid month.");
}
```

6 – Les opérateurs

Switch

Nous pouvons utiliser une instruction ***yield*** pour spécifier la valeur d'une expression switch.

- Pour les lignes simples, écrivez la valeur après le ->
- Pour les blocs, utilisez le mot-clé yield.
  - ✓ yield n'a une signification particulière que dans les expressions switch.
  - ✓ si une variable yield existe ailleurs, ce code ne sera pas interrompu ;

Tous les cas doivent générer une valeur.

```
int numberOfDays = switch(month) {
    case 1, 3, 5, 7, 8, 10, 12 -> 31;
    case 4, 6, 9, 11           -> 30;
    case 2 -> {
        if (!isLeapYear) {
            yield 28;
        } else
            yield 29;
    }
    default -> 0;
};
```

7 – Les méthodes

Déclaration

Les classes peuvent contenir des définitions de méthodes pour implémenter des comportements sur leurs instances (objets).

- Une méthode doit déclarer son type de retour ou utiliser le mot-clé void si une méthode n'a pas besoin de retourner une valeur.
- Les méthodes non-void doivent contenir une instruction return, qui doit retourner une valeur du type correspondant.
- Le nom de la méthode est généralement un verbe (comme "get" ou "set") écrit en minuscules, suivi de noms descriptifs.

```
package <package name>;
<access modifier> class <ClassName> {
  <access modifier> <return type> <method name>(<ParameterType> <parameterName>,
                                                 <ParameterType> <parameterName>) {

    return <value>;
  }
}
```

```
package demos.shop;
public class Product {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String newName) {
        name = newName;
    }
}
```

- Les modificateurs d'accès déterminent à partir de où la méthode peut être invoquée.
- Les méthodes peuvent décrire une liste de paramètres séparée par des virgules, entourée par des symboles "(" et ")".
- Le corps de la méthode est délimité par les symboles "{" et "}".

7 – Les méthodes

Déclaration

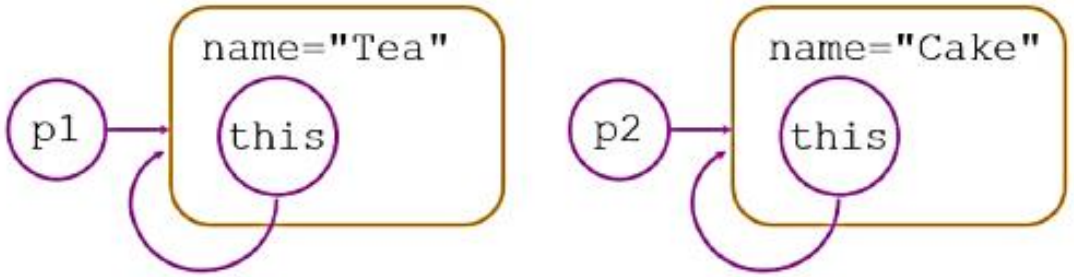
Les variables déclarées à l'intérieur des méthodes sont appelées des variables locales.

- Aucun modificateur d'accès ne peut être appliqué aux variables locales. (Elles ne sont de toute façon pas visibles en dehors de la méthode.)
- Les paramètres de méthode sont essentiellement des variables locales.
- La variable locale peut "masquer" la variable d'instance si leurs noms coïncident.
- Utilisez le mot-clé this (référence récursive à l'objet courant) pour faire référence à une instance, plutôt qu'à une variable locale.
- Les variables définies dans des blocs internes de code (délimités par les symboles "{" et "}") ne sont pas visibles en dehors de ces blocs.

7 – Les méthodes

Déclaration

```
Product p1 = new Product();
Product p2 = new Product();
p1.setName("Tea");
p2.setName("Cake");
```



- Note: In the example, variables p1 and p2 are referencing different products, while this is referencing the current one.

```
public class Product {
    private String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        if (name == null) {
            String dummy = "Unknown";
            return dummy;
        }
        return name;
    }
    public String consume() {
        String feedback = "Good!";
        return feedback;
    }
}
```

7 – Les méthodes

Variables locales

Il n'est pas nécessaire de décrire le type d'une variable si celui-ci peut être déduit sans ambiguïté du contexte.

- Déduire les types des variables locales avec l'initialisation.
- Pas besoin de déclarer explicitement le type de la variable locale si celui-ci peut être inféré à partir de la valeur assignée.
- Cette fonctionnalité est limitée à :
  - Les variables locales avec initialisation ;
  - Les index dans les boucles for améliorées ;
  - Les variables locales déclarées dans une boucle for
  - Un usage excessif peut réduire la lisibilité du code.

```
public void someOperation(int param) {
  var value1 = "Hello"; // infers String
  var value2 = param;   // infers int
}
```

7 – Les méthodes

Variables locales

- Les constantes représentent des données qui sont assignées une seule fois et ne peuvent pas être modifiées.
- Le mot-clé final est utilisé pour marquer une variable comme une constante ; une fois initialisée, elle ne peut plus être modifiée.
- Les variables d'instance marquées comme final doivent être initialisées soit immédiatement, soit via tous les constructeurs.
- Les variables locales et les paramètres peuvent également être marqués comme final.
- Toute tentative de réassigner une variable final entraînera une erreur de compilation.

```
public class Product {
  private final String name = "Tea";
  private final BigDecimal price = BigDecimal.ZERO;
  public BigDecimal getDiscount(final BigDecimal discount) {
    return price.multiply(discount);
  }
}
```

```
public class Shop {
  public static void main(String[] args) {
    Product p = new Product();
    BigDecimal percentage = BigDecimal.valueOf(0.2);
    final BigDecimal amount = p.getDiscount(percentage);
  }
}
```

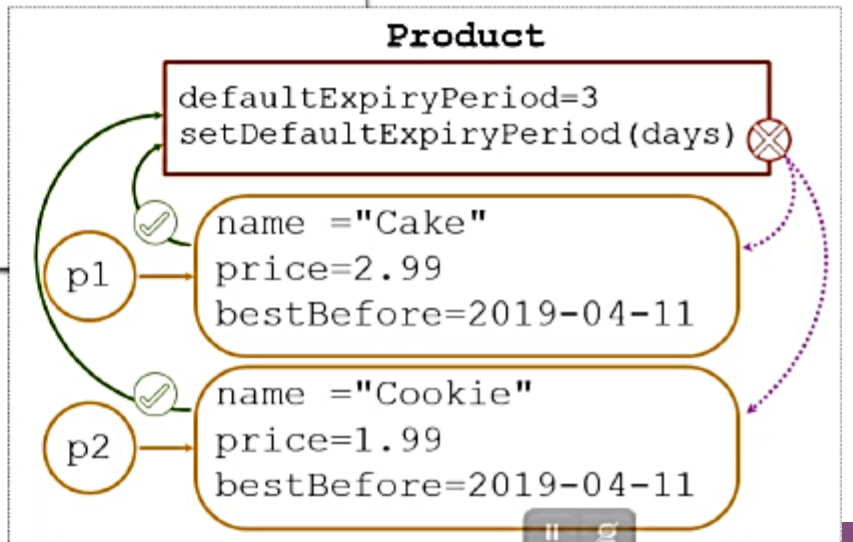
7 – Les méthodes

Static

- Chaque classe a son propre contexte mémoire.
- Le contexte mémoire d'une classe (également appelé contexte statique) est partagé par toutes les instances de cette classe.
- Le mot-clé static est utilisé pour marquer les variables ou méthodes qui appartiennent au contexte de la classe.
- Les objets peuvent accéder au contexte statique partagé.
- L'instance courante (this) n'a pas de sens dans le contexte statique.
- Toute tentative d'accès aux méthodes ou variables de l'instance courante depuis le contexte statique entraînera une erreur de compilation.

```
public class Product {
  private static Period defaultExpiryPeriod = Period.ofDays(3);
  private String name;
  private BigDecimal price;
  private LocalDate bestBefore;
  public static void setDefaultExpiryPeriod(int days){
    defaultExpiryPeriod = Period.ofDays(days);
    String name = this.name;
  }
}
```

```
Product p1 = new Product();
Product p2 = new Product();
```



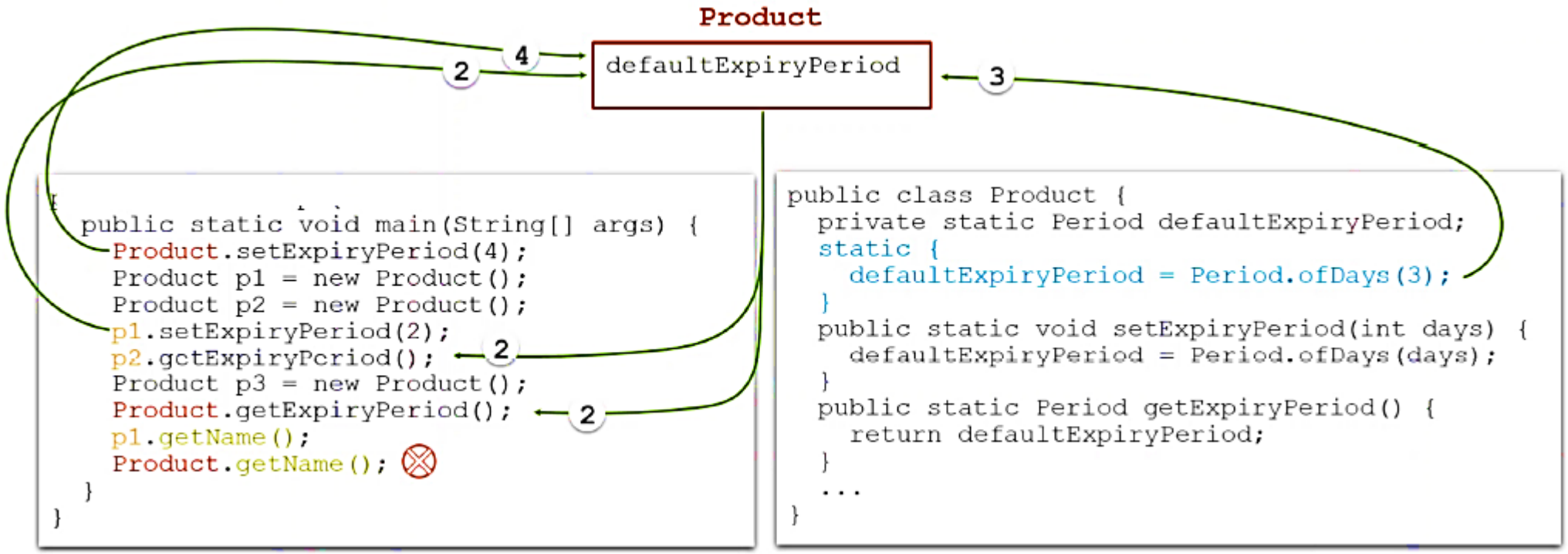
✣Reminder: Java Classes are types and Objects are their instances.

7 – Les méthodes

Static

Règles pour accéder au contexte statique :

- Il n'est pas nécessaire d'utiliser une référence d'objet (mais cela peut être fait) pour accéder au contexte statique.
- L'initialiseur statique s'exécute une seule fois, avant toute autre opération (lorsque la classe est chargée).
- Les variables d'instance et les méthodes ne sont pas accessibles à travers le contexte statique.



7 – Les méthodes

Static

- Les constantes partagées peuvent être définies en tant que variables static et final.
- Cela offre un moyen simple de définir des constantes globalement visibles.
- L'encapsulation (modificateurs d'accès private) n'est pas nécessaire car la valeur est en lecture seule.
- Une constante peut également être définie en tant que variable d'instance ou locale et, par conséquent, peut avoir une valeur d'initialisation différente pour chaque instance ou contexte de méthode.

```
public class Product {
  public static final int MAX_EXPIRY_PERIOD = 5;
  ...
}
```

```
public class Shop {
  public static void main(String[] args) {
    Period expiry = Period.days(Product.MAX_EXPIRY_PERIOD);
    ...
  }
}
```

7 – Les méthodes

Surcharge

- La surcharge de méthode permet de définir plusieurs versions d'une méthode au sein d'une même classe.
- Les méthodes surchargées suivent ces règles : deux ou plusieurs méthodes, dans la même classe, qui ont des noms identiques ➔ **doivent avoir un nombre différent, ou des types différents, ou un ordre différent de paramètres.**
- La surcharge est pratique : l'utilisateur n'a pas besoin d'apprendre différents noms de méthodes, il suffit de passer des paramètres différents.

```
public class Product {
  private BigDecimal price;
  private BigDecimal discount = BigDecimal.ZERO;
  public void setPrice(double price) {
    this.price = BigDecimal.valueOf(price);
  }
  public void setPrice(BigDecimal price) {
    this.price = price;
  }
  public void setPrice(BigDecimal price,
                       BigDecimal discount) {
    this.price = price;
    this.discount = discount;
  }
}
```

Methods cannot be overloaded:

- Using only different parameter names
- Using different return types

```
Product p = new Product();
p.setPrice(1.99);
p.setPrice(BigDecimal.valueOf(1.99));
p.setPrice(BigDecimal.valueOf(1.99),
           BigDecimal.valueOf(0.9));
```

7 – Les méthodes

varargs

- La fonctionnalité **vararg** permet de passer un nombre variable d'arguments du même type.
- Elle évite de créer trop de versions surchargées de la même méthode.
- Le paramètre **vararg** est traité comme un tableau, avec la constante de longueur indiquant le nombre de valeurs.
- Un indice entier (commençant à 0) est utilisé pour accéder aux éléments du tableau.
- Dans le cas où il y a d'autres paramètres dans la méthode, le paramètre **vararg** doit être défini en dernier.

```
Product p = new Product();
p.setFiscalDetails(1.99);
p.setFiscalDetails(1.99, 0.9, 0.1);
p.setFiscalDetails(1.99, 0.9);
```

```
public class Product {
  private BigDecimal price;
  private BigDecimal discount;
  private BigDecimal tax;
  public void setFiscalDetails(double... values) {
    switch (values.length) {
      case 3:
        tax = BigDecimal.valueOf(values[2]);
      case 2:
        discount = BigDecimal.valueOf(values[1]);
      case 1:
        price = BigDecimal.valueOf(values[0]);
    }
  }
}
```

hank

ou!

Hope you enjoyed this first chapter and feel excited to dive deeper into Java :)