**1) Les Types et les Définition en Java :**

Les types dans java sont deux type primitifs et wrapper class des classes enveloppes (Integer, Double ) , l’utilité change en fonction de l’utilisation les type primitifs dont généralement utilisé dans des condition des stockage du data vu que il prennent pas assez d’espace comme un objet de type wrapper par contre l’utilité des wrapper class consiste a les utilisé dans des generics contexte comme les arraylist, hashmap ….

l’autoboxing et l’unboxing permet de transformer automatiquement une variable de type primitif en un objet de type wrapper et l unboxing c’est l’inverse

String montexte = "10"; Integer monnombre = new Integer(montexte); int i = monnombre.intValue();

**2) Le Safe Casting**:

pour caster une variable de type predifnit vers un autre type peut engendré des exceptions tel que ClassCastException pour eviter cette exception on peut utiliser instanceof

if (animal instanceof Cat) { Cat cat = (Cat) animal; cat.meow();}

else if (animal instanceof Dog) { Dog dog = (Dog) animal; dog.woof(); }

**3) La Gestion Avancée des Classes et Objets**

structure des classes efficace : Access Modifier

**Abstract**: il y’a au moins une methode abstraites et ne peut pas etre instancié

**Final**: La class ne peut pas etre modifier , pas de child class

**Private**: la class n’est accessible qu’a partir le fichier

**Public**: La classe est accessible partout

**Access modifier for methods :**

**public :** la methode est accessible aux autres method de autres classes

**private**: l usage est reserve aux autre methodes de la meme classe

**protected**: la méthode ne peut être invoquée que par des méthodes de la classe ou de ses sous-classes

**fina l**:la méthode ne peut être modifiée (redéfinition lors de l'héritage interdite)

**static**: la méthode appartient simultanément à tous les objets de la classe (comme une constante déclarée à l'intérieur de la classe).

**synchronized**:

**native**:

**la surcharge des methods**:permet de définir plusieurs fois une même méthode avec des arguments différents

**4) Les Génériques :** sont un ensemble de caractéristique du langage liées a la définition

5) **Api Stream :**

Un stream(flux) est une représentation d’une séquence sur laquelle il est possible d’appliquer des opérations. Cette API a deux principales intérêts

1 : Elle permet d’effectuer les opérations sur une séquence sans utiliser de structure de boucle.

2 : Les opérations sur les streams sont réalisées en flux (d’où leur nom) ce qui limite l’empreinte mémoire nécessaire

La collecte permet de créer un nouvelle collection à partir d’un stream. Pour cela, il faut fournir une implémentation de l’interface [Collector](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/stream/Collector.html). 

**List<String> liste = Arrays.asList("une chaine", "une autre chaine", "encore une chaine");**

**List<String> autreListe = liste.stream().collect(Collectors.toList());**

## Le filtrage

Une opération courante sur un stream consiste à appliquer un filtre pour éliminer une partie de ses éléments. Pour, cela on peut utiliser la méthode [Stream.filter](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/stream/Stream.html" \l "filter(java.util.function.Predicate)).

**List<Voiture> liste = Arrays.asList(new Voiture("citroen"), new Voiture("audi"), new Voiture("citroen"));**

**List<Voiture> sansCitroen = liste.stream() .filter(v -> !v.getMarque().equals("citroen")) .collect(Collectors.toList());**

**System.out.println(sansCitroen.size()); // 1**

**5) Design Pattern :**

**Factory :**La fabrique permet de créer un objet dont le type dépend du contexte  cet objet fait partie d'un ensemble de sous-classes. L'objet retourné par la fabrique est donc toujours du type de la classe mère mais grâce au polymorphisme les traitements exécutés sont ceux de l'instance créée. Une Factory est une classe qui instancie des objets d'une autre classe