

Voici le résumé structuré **non exhaustif** (généré par NotebookLM) des notions et compétences à maîtriser pour les deux premières séquences (TP1/Cours1 et TP2/TD2/Cours2) :

1. Concepts fondamentaux de l'Orienté Objet

- **Classe vs Objet** : Une classe est le modèle de code (ex: `Cercle`), tandis qu'un objet (ou instance) est la concrétisation en mémoire créée via l'instruction `new`.
- **Encapsulation & Visibilité** : Les attributs sont déclarés en `private` pour être protégés, alors que les méthodes sont en général `public`. La protection `private` s'applique au niveau de la classe : une méthode peut accéder aux attributs privés d'un autre objet s'il appartient à la même classe.
- **Utilisation de `this`** : Ce mot-clé désigne l'objet courant sur lequel la méthode est exécutée (`this.aAttribut` ou `this.methode()`).

2. Organisation du code & Typologie des variables

- **Structure d'une classe** : L'ordre préconisé est : 1) Attributs, 2) Constructeurs, 3) Autres méthodes.
- **Les 3 sortes de variables** :
 - **Attributs** : Caractéristiques de l'objet, déclarées en début de classe.
 - **Paramètres** : Informations reçues par une méthode entre ses parenthèses (déclarés `final`).
 - **Variables locales** : Variables temporaires créées au sein d'un bloc `{ }` dans une méthode.
- **Les 2 sortes de types** :
 - **Types primitifs** (`int`, `boolean`, `byte`, `short`, `long`) stockant directement une valeur.
 - **Types objets / références** (`String`, `Cercle`, etc.) stockant l'adresse d'un objet en mémoire.

3. Les Méthodes et Constructeurs

- **Les 3 sortes de méthodes** :
 - **Procédure** : Effectue des actions sans retourner de valeur (`void`).
 - **Fonction** : Calcule et retourne toujours une valeur unique via l'instruction `return`.
 - **Constructeur** : Méthode appelée automatiquement lors du `new` pour initialiser les attributs ; elle porte exactement le nom de la classe, sans type de retour ni `void`.

- **Constructeur par défaut & naturel** : Le constructeur par défaut est sans paramètre, tandis que le constructeur naturel reçoit des paramètres pour initialiser tous les attributs.
- **Surcharge & Chaînage (`this(...)`)** :
 - **Surcharge** : Possibilité de définir plusieurs méthodes ou constructeurs de même nom avec des signatures différentes.
 - **Chaînage** : Pour éviter la duplication de code, un constructeur peut appeler un autre constructeur de la même classe à la première ligne avec `this(...)`.
- **Accesseurs & Modificateurs** : Utilisation systématique de fonctions getters (`get...()`) pour lire un attribut et de procédures setters (`set...()`) pour le modifier sous contrôle.
- **Méthodes classiques** : Conception de méthodes comme `toString()` pour la conversion en texte ou `egal()` pour tester l'égalité entre deux objets.

4. Algorithmique & Gestion des erreurs

- **Récurtivité** : Méthode qui s'appelle elle-même avec un paramètre modifié. Elle nécessite impérativement une **condition d'arrêt (cas de base)** pour éviter une boucle infinie générant l'erreur `StackOverflowError`.
- **Erreur `NullPointerException`** : Déclenchée lorsqu'on tente d'appeler une méthode ou d'accéder à un attribut sur une variable valant `null` (référence absente).
- **Expressions vs Instructions** : Une expression calcule une valeur (ex: `a == b`), tandis qu'une instruction exécute une action (ex: `a = b;` ou `return x;`).

5. Outils, Conventions & Javadoc

- **Environnement BlueJ** : Compilation (`.java` vers `.class`), inspection d'objets, visualisation du diagramme de dépendances (`uses`) et utilisation du CodePad.
- **Guide de style & Conventions de nommage** :
 - Nom de classe avec une majuscule initiale (`Cercle`).
 - Préfixe **a** pour les attributs (ex: `aDiametre`).
 - Préfixe **p** pour les paramètres (ex: `pDiametre`).
 - Préfixe **v** pour les variables locales (ex: `vNombre`).
- **Javadoc** : Rédaction des commentaires structurés (`/** ... */`) avec les balises `@param`, `@return`, `@author` et `@version` pour générer la documentation automatique