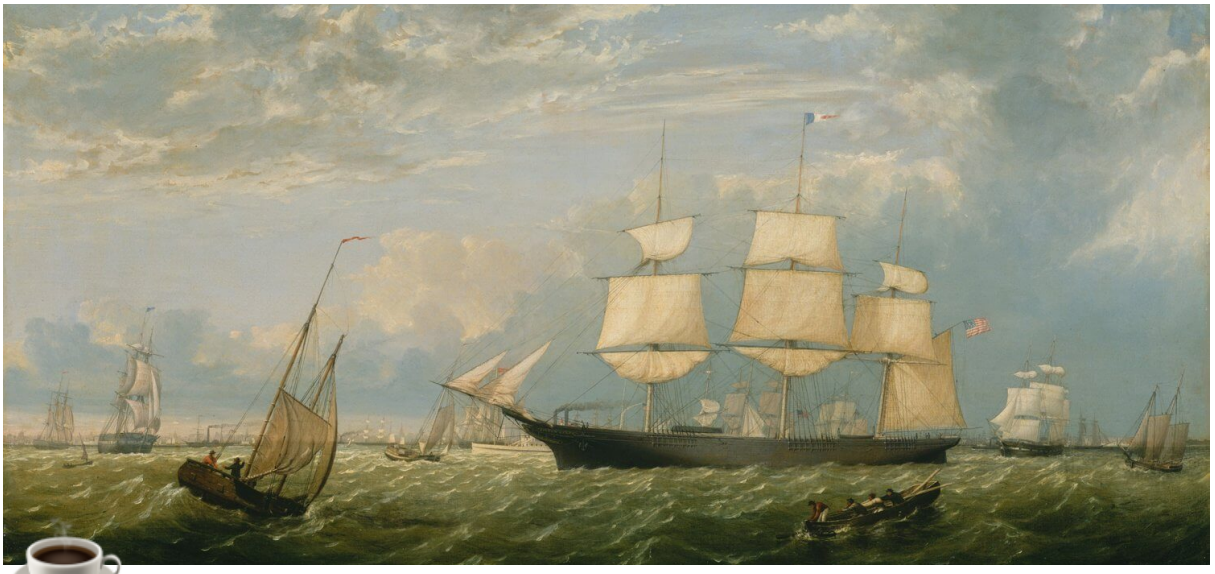

Résumé JAVA

Constructeurs

1. **C'est quoi un constructeur ?**
    - Une méthode spéciale utilisée pour initialiser un objet lors de sa création.
    - Même nom que la classe, sans type de retour.

```
class Animal {
    String nom;
    Animal(String nom) { this.nom = nom; }
}
```

2. **Surcharge des constructeurs** :
    - Plusieurs constructeurs dans une classe avec des signatures différentes.

```
Animal {
    String nom; int age;
    Animal(String nom) { this.nom = nom; }
    Animal(String nom, int age) { this.nom = nom; this.a
```

```
ge = age; }
}
```

3. **Chaînage des constructeurs** :
   - Un constructeur appelle un autre à l'aide de `this()`.

```
class Animal {
    String nom; int age;
    Animal(String nom) { this(nom, 0); }
    Animal(String nom, int age) { this.nom = nom; this.a
ge = age; }
}
```

Modificateurs d'accès et de non-accès

1. Modificateurs d'accès

- Contrôlent la visibilité des membres d'une classe.

Modificateur	Visibilité	Exemple
`public`	Accessible partout.	`public int x;`
`private`	Accessible uniquement dans la classe.	`private int x;`
`protected`	Accessible dans le package et par héritage.	`protected int x;`
`default`	(Pas de mot-clé) : Accessible dans le package.	`int x;`

Différence : `private` protège au maximum, `public` rend accessible partout.

Exemple :

```
class Exemple {
    public int pubVar = 1;
    private int privVar = 2;
    protected int protVar = 3;
    int defVar = 4;
}
```

2. Modificateurs de non-accès

- Ajoutent des comportements spécifiques.

Modificateur	Description	Exemple
`final`	Empêche la modification (variable, méthode, classe).	`final int x = 5;`
`static`	Appartient à la classe, pas à l'instance.	`static int compteur;`
`abstract`	Définit une méthode/classe abstraite.	`abstract class Forme { ... }`

Exemple pour chaque :

- `final` :

```
final class Constante { final int x = 10; }
```

- `static` :

```
lass MathUtils { static int addition(int a, int b) { ret
urn a + b; } }
```

- `abstract` :

```
bstract class Forme { abstract void dessiner(); }
class Cercle extends Forme { void dessiner() { System.ou
t.println("Cercle"); } }
```

Polymorphisme

1. **Surcharge de méthode (Overloading)** :
   - Même nom, paramètres différents (compile-time).

```
class MathUtils {
    int addition(int a, int b) { return a + b; }
    double addition(double a, double b) { return a + b;
}
}
```

2. **Redéfinition de méthode (Overriding)** :

- Même signature, comportement différent dans une classe fille (run-time).

```
Animal { void son() { System.out.println("Son génériqu
e"); } }
class Chien extends Animal { void son() { System.out.pri
ntln("Aboiement"); } }
```

Différence :

- La surcharge se fait dans la même classe (compile-time).
- La redéfinition se fait entre parent et enfant (run-time).

Héritage

- Une classe enfant hérite des attributs et méthodes d'une classe parent.
- Utilise le mot-clé `extends`.

Exemple :

```
class Animal {
    void dormir() { System.out.println("Dort"); }
}
class Chien extends Animal {
    void aboyer() { System.out.println("Aboie"); }
}
```

Différence avec encapsulation :

- Héritage partage les attributs/méthodes.
- Encapsulation protège les données via des modificateurs.

Encapsulation

- Restriction d’accès aux données avec des **getters** et **setters**.
- Protège les données et contrôle leur accès.

Exemple :

```
class Personne {
    private String nom;
    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }
}
```

Différence :

- L'encapsulation se concentre sur la sécurité des données.
- L'héritage, en revanche, partage les comportements entre classes.

Abstraction

1. Classe abstraite en Java

- Une classe abstraite est une classe qui **ne peut pas être instanciée**.
- Peut contenir des **méthodes abstraites** (sans corps) et **méthodes concrètes** (avec corps).
- Utilisée pour fournir une structure de base aux classes dérivées.
- Utilise le mot-clé `abstract`.

Exemple :

```
java
Copy code
abstract class Animal {
    abstract void son(); // Méthode abstraite
    void manger() { System.out.println("Mange"); } // Métho
de concrète
}
class Chien extends Animal {
    void son() { System.out.println("Aboie"); }
}
```

Points clés :

- Une classe enfant **doit implémenter toutes les méthodes abstraites** de la classe parent.
- Peut contenir des variables ou méthodes `final` ou `static`.

---

2. Interfaces en Java

- Une interface est une collection de **méthodes abstraites** qui définissent un contrat que les classes doivent respecter.
- Depuis Java 8, les interfaces peuvent inclure des **méthodes par défaut** et **méthodes statiques**.
- Utilise le mot-clé `interface`.

Exemple :

```
java
Copy code
interface Volant {
    void voler(); // Méthode abstraite
}
class Avion implements Volant {
    public void voler() { System.out.println("Avion vole");
}
}
```

Différences entre Classe Abstraite et Interface :

Aspect	Classe Abstraite	Interface
Héritage	Une seule classe par héritage.	Plusieurs interfaces peuvent être implémentées.
Méthodes	Peut contenir des méthodes concrètes.	Méthodes abstraites (par défaut concrètes depuis Java 8).
Variables	Peut avoir des attributs.	Les attributs sont `final` et `static` par défaut.
Mots-clés utilisés	`abstract`	`interface`

Résumé des usages :

- **Classe abstraite** : Quand vous avez besoin d’une structure commune avec des comportements partagés.
- **Interface** : Quand vous définissez un contrat ou des comportements à partager entre des classes sans relation hiérarchique.

BY yours truly amen