Exercice : Implémenter le polymorphisme avec des classes d'animaux

Objectifs:

- Comprendre et implémenter le polymorphisme en Java.
- Entraînez-vous à utiliser des classes abstraites et des interfaces.
- Démontrer le remplacement de méthode et la répartition dynamique de méthodes.

Instructions:

1. **Créez une classe abstraite `Animal` :**

- Déclarez une méthode abstraite `makeSound()`.
- Déclarez une méthode concrète `eat()` qui affiche un message disant que l'animal mange.

2. **Créez des sous-classes qui étendent `Animal` :**

- `Dog` : Implémente la méthode `makeSound()` pour imprimer "Woof".
- `Cat` : Implémente la méthode `makeSound()` pour imprimer "Meow".
- `Cow` : Implémente la méthode `makeSound()` pour imprimer "Moo".

3. **Créez une classe principale `Farm` :**

- Dans la méthode `main`, créez un tableau d'objets `Animal`.
- Remplissez le tableau avec des instances de « Chien », « Chat » et « Vache ».
- Parcourez le tableau et pour chaque animal, appelez les méthodes `makeSound()` et `eat()`.

4. **Bonus** : Implémentez une interface `Pet` avec une méthode `play()`.

- Demandez à `Dog` et `Cat` d'implémenter l'interface `Pet`.
- Dans la classe `Farm`, vérifiez si un animal est une instance de `Pet` et appelez la méthode `play()` si c'est le cas.

Exemple de structure de code :

1. **Animal.java**

```
public abstract class Animal {
    public abstract void makeSound();
```

```
    public void eat() {
        System.out.println("This animal is eating.");
    }
}
```

---

2. **Dog.java**

```
public class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof");
    }
}
```

---

3. **Cat.java**

```
public class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Meow");
    }
}
```

---

4. **Cow.java**

```
public class Cow extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Moo");
    }
}
```

---

5. **Farm.java**

```
public class Farm {
    public static void main(String[] args) {
        Animal[] animals = {new Dog(), new Cat(), new Cow()};
```

```
        for (Animal animal : animals) {
            animal.makeSound();
            animal.eat();
        }
    }
}
```

---

6. **Bonus - Pet.java**

```
public interface Pet {
    void play();
}
```

---

7. **Dog.java (with Pet)**

```
public class Dog extends Animal implements Pet {
    @Override
    public void makeSound() {
        System.out.println("Woof");
    }

    @Override
    public void play() {
        System.out.println("Dog is playing.");
    }
}
```

---

8. **Cat.java (with Pet)**

```
public class Cat extends Animal implements Pet {
    @Override
    public void makeSound() {
        System.out.println("Meow");
    }

    @Override
    public void play() {
        System.out.println("Cat is playing.");
```

```
    }
}
```

---

9. **Farm.java (with Pet check)**

```java
public class Farm {
    public static void main(String[] args) {
        Animal[] animals = {new Dog(), new Cat(), new Cow()};

        for (Animal animal : animals) {
            animal.makeSound();
            animal.eat();

            if (animal instanceof Pet) {
                ((Pet) animal).play();
            }
        }
    }
}
```

Explication:

- Le **polymorphisme** est démontré par la possibilité d'appeler les méthodes `makeSound()` et `eat()` sur différents types d'objets `Animal` sans connaître leurs types spécifiques au moment de la compilation.
- **La classe abstraite `Animal`** définit l'interface commune à tous les animaux.
- **Le remplacement de méthode** est utilisé dans `Dog`, `Cat` et `Cow` pour fournir des implémentations spécifiques de la méthode `makeSound()`.
- **La répartition dynamique des méthodes** garantit que la méthode `makeSound()` correcte est appelée en fonction du type réel de l'objet au moment de l'exécution.
- **L'interface `Pet`** et l'utilisation de `instanceof` dans la classe `Farm` démontrent comment ajouter un comportement supplémentaire à des sous-classes spécifiques.