Exercice 1: Classes de Base et Dérivées

Objectif

Créer une classe de base `Animal` et une classe dérivée `Dog` qui hérite de `Animal`.

Instructions

1. Créez une classe `Animal` avec les attributs `name` et `age`, et une méthode `makeSound()` qui affiche un son générique.
2. Créez une classe `Dog` qui hérite de `Animal` et ajoute un attribut `breed`. Overridez la méthode `makeSound()` pour afficher "Woof".

Solution

```
class Animal {
    String name;
    int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void makeSound() {
        System.out.println("Some generic animal sound");
    }
}

class Dog extends Animal {
    String breed;

    Dog(String name, int age, String breed) {
        super(name, age);
        this.breed = breed;
    }

    @Override
    void makeSound() {
        System.out.println("Woof");
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal("Generic Animal", 5);
        myAnimal.makeSound();

        Dog myDog = new Dog("Buddy", 3, "Golden Retriever");
        myDog.makeSound();
    }
}
```

Exercice 2: Utilisation de `super`

Objectif

Utiliser le mot-clé `super` pour accéder à la méthode de la classe parente.

Instructions

1. Ajoutez une méthode `displayInfo()` dans la classe `Animal` pour afficher les informations de base.
2. Overridez `displayInfo()` dans la classe `Dog` pour inclure le `breed` et utilisez `super` pour réutiliser la méthode de la classe parente.

Solution

```
class Animal {
    String name;
    int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void makeSound() {
        System.out.println("Some generic animal sound");
    }
```

```
    void displayInfo() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
}

class Dog extends Animal {
    String breed;

    Dog(String name, int age, String breed) {
        super(name, age);
        this.breed = breed;
    }

    @Override
    void makeSound() {
        System.out.println("Woof");
    }

    @Override
    void displayInfo() {
        super.displayInfo();
        System.out.println("Breed: " + breed);
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog("Buddy", 3, "Golden Retriever");
        myDog.displayInfo();
    }
}
```

Exercice 3: Héritage et Constructeurs

Objectif

Comprendre comment les constructeurs fonctionnent avec l'héritage.

Instructions

1. Créez une classe de base `Vehicle` avec un constructeur qui initialise

`make` et `model`.

2. Créez une classe dérivée `Car` avec un constructeur qui initialise `make`, `model` et `year`.
3. Appelez le constructeur de la classe parente depuis le constructeur de la classe dérivée en utilisant `super`.

Solution

```
class Vehicle {
    String make;
    String model;

    Vehicle(String make, String model) {
        this.make = make;
        this.model = model;
    }

    void displayInfo() {
        System.out.println("Make: " + make + ", Model: " + model);
    }
}

class Car extends Vehicle {
    int year;

    Car(String make, String model, int year) {
        super(make, model);
        this.year = year;
    }

    @Override
    void displayInfo() {
        super.displayInfo();
        System.out.println("Year: " + year);
    }
}

public class Main {
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", "Corolla", 2020);
```

```
        myCar.displayInfo();
    }
}
```

Exercice 4: Polymorphisme

Objectif

Utiliser le polymorphisme avec l'héritage.

Instructions

1. Créez une méthode `printSound()` qui accepte un objet `Animal` et appelle sa méthode `makeSound()`.
2. Passez différents objets `Animal` et `Dog` à cette méthode pour observer le comportement polymorphique.

Solution

```
class Animal {
    String name;
    int age;

    Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void makeSound() {
        System.out.println("Some generic animal sound");
    }
}

class Dog extends Animal {
    String breed;

    Dog(String name, int age, String breed) {
        super(name, age);
        this.breed = breed;
    }
```

```
    @Override
    void makeSound() {
        System.out.println("Woof");
    }
}

public class Main {
    public static void printSound(Animal animal) {
        animal.makeSound();
    }

    public static void main(String[] args) {
        Animal myAnimal = new Animal("Generic Animal", 5);
        Dog myDog = new Dog("Buddy", 3, "Golden Retriever");

        printSound(myAnimal); // Some generic animal sound
        printSound(myDog);    // Woof
    }
}
```

Exercice 5: Classe Abstraite

Objectif

Comprendre l'utilisation des classes abstraites et des méthodes abstraites.

Instructions

1. Créez une classe abstraite `Shape` avec une méthode abstraite `calculateArea()`.
2. Créez des classes `Circle` et `Rectangle` qui étendent `Shape` et implémentent `calculateArea()`.

Solution

```
abstract class Shape {
    abstract double calculateArea();
}

class Circle extends Shape {
    double radius;
```

```
    Circle(double radius) {
        this.radius = radius;
    }

    @Override
    double calculateArea() {
        return Math.PI * radius * radius;
    }
}

class Rectangle extends Shape {
    double width, height;

    Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    double calculateArea() {
        return width * height;
    }
}

public class Main {
    public static void main(String[] args) {
        Shape circle = new Circle(5);
        Shape rectangle = new Rectangle(4, 6);

        System.out.println("Circle area: " + circle.calculateArea());
        System.out.println("Rectangle area: " + rectangle.calculateArea());
    }
}
```