Programmation orientée objet : Encapsulation et héritage - Quiz

https://github.com/heig-vd-progim-course/heig-vd-progim2-course

Visualiser le contenu complet sur GitHub à cette adresse.

V. Guidoux, avec l'aide de GitHub Copilot.

Ce travail est sous licence CC BY-SA 4.0.

Plus de détails sur GitHub

Cette présentation est un quiz pour tester ses connaissances sur le chapitre en cours. Pour plus de détails, consultez le contenu complet sur GitHub.

Question 1 - Donnée

Complétion : Modificateurs d'accès

Quel modificateur permet de rendre un attribut accessible uniquement depuis la classe elle-même ?

- **A.** `public`
- **B.** `private`
- **C.** `protected`
- **D.** `default`

Question 1 - Réponse

Réponse correcte : B `private`

```
class BankAccount {
    private double balance;  // Accessible uniquement depuis BankAccount
}
```

- **private** : accès uniquement dans la classe
- **public** : accès depuis partout
- **protected** : accès dans la classe et ses sous-classes
- **default** : accès dans le même package

Question 2 - Donnée

Complétion : Getters et setters

Comment appelle-t-on une méthode qui retourne la valeur d'un attribut privé ?

- **A.** Un mutateur (setter)
- **B.** Un constructeur
- **C.** Un accesseur (getter)
- **D.** Un modificateur

Question 2 - Réponse

Réponse correcte : C - Un accesseur (getter)

```
class Person {
    private String name;

    // Accesseur (getter)
    public String getName() {
        return name;
    }

    // Mutateur (setter)
    public void setName(String name) {
        this.name = name;
    }
}
```

Question 3 - Donnée

Quel code valide correctement l'âge dans un setter ?

```
// Option A
public void setAge(int age) {
    this.age = age;
}

// Option B
public void setAge(int age) {
    if (age > 0) {
        this.age = age;
    }
}
```

Question 3 - Réponse

Réponse correcte : B - Validation avec condition

```
class Person {
    private int age;

    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
    }
}
```

Un setter permet de valider les données avant de les affecter.

Question 4 - Donnée

Complétion : Le mot-clé final

Que signifie le mot-clé `final` sur une variable ?

```
final double PI = 3.14159;
```

- **A.** La variable peut changer une seule fois
- **B.** La variable ne peut jamais être modifiée
- **C.** La variable est privée
- **D.** La variable est publique

Question 4 - Réponse

Réponse correcte : B - La variable ne peut jamais être modifiée

```
class Circle {
    final double PI = 3.14159;  // Constante

    void test() {
        PI = 3.14;  // ERREUR : ne compile pas
    }
}
```

Explications : `final` crée une constante qui ne peut plus être modifiée après son initialisation.

Question 6 - Donnée

Que fait le mot-clé `super` dans ce constructeur ?

```
class Vehicle {
    String brand;
    Vehicle(String brand) {
        this.brand = brand;
    }
}
class Car extends Vehicle {
    String model;
    Car(String brand, String model) {
        super(brand);
        this.model = model;
    }
}
```

Question 6 - Réponse

Réponse correcte : Appelle le constructeur de la classe parent

Explications :

- `super()` appelle le constructeur de la classe parent
- Doit être la **première instruction** du constructeur

Question 7 - Donnée

Quelle différence entre `private` et `protected` ?

```
class Parent {
    private int x = 10;
    protected int y = 20;
}

class Child extends Parent {
    void display() {
        // System.out.println(x);  // ?
        // System.out.println(y);  // ?
    }
}
```

Question 7 - Réponse

`private` : inaccessible aux sous-classes

`protected` : accessible aux sous-classes

```
class Parent {
    private int x = 10;      // Inaccessible dans Child
    protected int y = 20;    // Accessible dans Child
}
class Child extends Parent {
    void display() {
        // System.out.println(x);  // ERREUR
        System.out.println(y);     // OK : affiche 20
    }
}
```

Question 8 - Donnée

Quelle est la différence entre ces deux classes ?

```
// Classe A
abstract class Animal {
    abstract void makeSound();
}

// Classe B
class Dog {
    void makeSound() {
        System.out.println("Woof");
    }
}
```

Question 8 - Réponse

```
abstract class Animal {  // Classe ABSTRAITE
    abstract void makeSound();  // Méthode abstraite (pas d'implémentation)
}

class Dog extends Animal {  // Classe CONCRÈTE
    void makeSound() {  // Implémentation de la méthode abstraite
        System.out.println("Woof");
    }
}

Animal a = new Animal();  // ERREUR : classe abstraite
Dog d = new Dog();  // OK : classe concrète
```

Question 9 - Donnée

Quelle est la différence entre ces deux méthodes `display` ?

```
class Parent {
    void display() {
        System.out.println("Parent");
    }
}
class Child extends Parent {
    void display() {                // Méthode 1
        System.out.println("Child");
    }
    void display(String msg) {      // Méthode 2
        System.out.println(msg);
    }
}
```

Question 9 - Réponse

Méthode 1 = redéfinition, Méthode 2 = surcharge

```
class Child extends Parent {
    @Override
    void display() {                    // REDÉFINITION
        System.out.println("Child");  // Même signature que Parent
    }

    void display(String msg) {      // SURCHARGE
        System.out.println(msg);    // Signature différente
    }
}
```

Explications :

Question 10 - Donnée

Ce code ne compile pas. Pourquoi ?

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

class Circle extends Shape {
    double radius;

    double getArea() {
        return Math.PI * radius * radius;
    }
}
```

Question 10 - Réponse

Réponse correcte : La classe doit être déclarée `abstract`

```
// ERREUR : méthode abstraite dans une classe non-abstraite
class Shape {
    abstract double getArea();  // ERREUR
}

// ---

// CORRECT : classe abstraite avec méthode abstraite
abstract class Shape {
    abstract double getArea();
}
```

Question 11 - Donnée

Quel modificateur empêche une méthode d'être redéfinie dans une sous-classe ?

```
class Vehicle {
    ______ void start() {
        System.out.println("Starting...");
    }
}
class Car extends Vehicle {
    void start() {  // Devrait être impossible
        System.out.println("Car starting...");
    }
}
```

Question 11 - Réponse

Réponse correcte : `final`

```
class Vehicle {
    final void start() {  // Ne peut pas être redéfinie
        System.out.println("Starting...");
    }
}
class Car extends Vehicle {
    @Override
    void start() {  // ERREUR : ne compile pas
        System.out.println("Car starting...");
    }
}
```

Question 12 - Donnée

Combien de classes peuvent hériter directement de `Animal` ?

```
abstract class Animal { }

class Dog extends Animal { }
class Cat extends Animal { }
class Bird extends Animal { }
```

- **A.** Aucune limite
- **B.** Maximum 3 classes
- **C.** Une seule classe
- **D.** Maximum 10 classes

Question 12 - Réponse

Réponse correcte : A - Aucune limite

```
abstract class Animal { }

class Dog extends Animal { }
class Cat extends Animal { }
class Bird extends Animal { }
class Fish extends Animal { }
// ... autant de sous-classes que nécessaire
```

Questions

Est-ce que vous avez des questions sur le quiz ou le contenu ?

Sources

- Illustration principale par Markus Spiske sur Unsplash