Problema diamantului (Diamond Problem) apare în general în limbaje de programare care permit **moștenirea multiplă** (o clasă care moștenește din mai multe clase de bază), cum ar fi C++. În aceste limbaje, problema apare atunci când o clasă derivată moștenește două clase care la rândul lor moștenesc o clasă comună. Acest lucru creează ambiguitate deoarece există două căi diferite prin care membrii clasei de bază pot fi moșteniți în clasa derivată.

**Cum funcționează problema diamantului**

Pentru a ilustra problema, să luăm în considerare următorul scenariu:

1. **Clasa A**: Este o superclasă comună.
2. **Clasa B și Clasa C**: Ambele moștenesc A.
3. **Clasa D**: Moștenește atât B, cât și C.

În această situație, dacă A are un atribut sau o metodă, D moștenește două copii ale acestora: una prin B și alta prin C. Astfel, dacă D încearcă să acceseze acel atribut sau să apeleze acea metodă, va exista o ambiguitate cu privire la care dintre copiile moștenite ar trebui să fie utilizată. Aceasta este **problema diamantului**.

**Moștenirea multiplă și Java**

**Java** nu permite moștenirea multiplă de clase pentru a evita această problemă. Cu alte cuvinte, o clasă în Java poate moșteni direct doar o singură clasă. Aceasta înseamnă că structura diamantei așa cum a fost descrisă mai sus nu poate apărea în Java atunci când vorbim despre moștenire de clase.

**Cum abordează Java problema diamantului**

Deși Java nu permite moștenirea multiplă de clase, permite totuși moștenirea multiplă prin **interfețe**. Atunci când o clasă implementează mai multe interfețe, iar aceste interfețe conțin metode cu aceeași semnătură, nu apare problema diamantului în sensul clasic. Cu toate acestea, poate apărea o situație similară dacă acele interfețe au metode implicite (default methods).



Exercitii:

1.

a **Definirea unei clase de bază Shape**:

* Clasa trebuie să conțină două atribute: color (de tip String) și filled (de tip boolean).
* Clasa trebuie să aibă un constructor care inițializează aceste atribute.
* Include o metodă displayDetails() care afișează valorile acestor atribute.

b **Definirea unei clase derivate Circle**:

* Clasa Circle trebuie să moștenească clasa Shape.
* Adaugă un atribut suplimentar radius (de tip double).
* Clasa trebuie să aibă un constructor care primește color, filled, și radius, și apelează constructorul din clasa Shape pentru a inițializa atributele moștenite.
* Suprascrie metoda displayDetails() pentru a afișa și valoarea atributului radius.

c **Definirea unei clase derivate Rectangle**:

* Clasa Rectangle trebuie să moștenească clasa Shape.
* Adaugă două atribute suplimentare: width și height (ambele de tip double).
* Clasa trebuie să aibă un constructor care primește color, filled, width, și height, și apelează constructorul din clasa Shape pentru a inițializa atributele moștenite.
* Suprascrie metoda displayDetails() pentru a afișa și valorile atributelor width și height.

d **În clasa principală Main**:

* Creează un obiect de tip Circle și un obiect de tip Rectangle.
* Afișează detaliile fiecărui obiect folosind metoda displayDetails().

2.

a **Clasa de bază Vehicle**:

* Această clasă va avea un atribut fuelType (de tip String), care reprezintă tipul de combustibil al vehiculului.
* Va avea o metodă startEngine() care afișează un mesaj generic că motorul vehiculului a pornit.

b **Clasa derivată Car**:

* Va moșteni clasa Vehicle și va avea un atribut suplimentar numberOfDoors (de tip int).
* Va suprascrie metoda startEngine() pentru a afișa un mesaj specific unui autoturism.

c **Clasa derivată Motorcycle**:

* Va moșteni clasa Vehicle și va avea un atribut suplimentar hasSidecar (de tip boolean).
* Va suprascrie metoda startEngine() pentru a afișa un mesaj specific unei motociclete.

3. Write a Java program to create a class called Animal with a method called makeSound(). Create a subclass called Cat that overrides the makeSound() method to bark.

4. Write a Java program to create a class called Employee with methods called work() and getSalary(). Create a subclass called HRManager that overrides the work() method and adds a new method called addEmployee().

5. Write a Java program that creates a class hierarchy for employees of a company. The base class should be Employee, with subclasses Manager, Developer, and Programmer. Each subclass should have properties such as name, address, salary, and job title. Implement methods for calculating bonuses, generating performance reports, and managing projects.

6. Write a method to count all words in a string

7. Write a program that contains a class Car that extends a class Vehicle. Display the model, brand and the sound of the car object.

8. Write a method to count all vowels in a string.

Link-uri utile:

* <https://www.tutorjoes.in/java_programming_tutorial/inheritance_exercise_programs_in_java#google_vignette>
* <https://www.w3resource.com/java-exercises/index-inheritance.php>
* <https://www.youtube.com/watch?v=ilm7ZGHmlUw>