Q1:Can two interfaces mutually extend each other? Why or why not?

**لا، لا يمكن لواجهتين أن تمد كل منهما الأخرى في Java. السبب في ذلك هو أنه سيؤدي إلى إنشاء تبعية دائرية، وهو أمر غير مسموح به في Java. عندما تمد واجهة واجهة أخرى، فإنها ترث جميع طرق الواجهة الأم. إذا كانت واجهتان تمد كل منهما الأخرى، فسيؤدي ذلك إلى حلقة لا نهائية من الوراثة، وهو ما لا يمكن لمترجم Java حله.**

====================================================

**Q2** : What are some potential efficiency disadvantages of having very deep inheritance trees, that is, a large set of classes, A, B, C, and so on, such that B extends A, C extends B, D extends C, etc.?

**وجود شجرة وراثة عميقة يمكن أن يؤدي إلى عدة عيوب من حيث الكفاءة:**

1. **زيادة التعقيد: شجرة الوراثة العميقة يمكن أن تجعل قاعدة الشيفرة أكثر تعقيدًا وصعوبة في الفهم. هذا يمكن أن يؤدي إلى زيادة الوقت اللازم للتطوير والصيانة حيث يحتاج المطورون إلى التنقل عبر طبقات متعددة من الوراثة لفهم سلوك الفئة.**
2. **زيادة الحمل على الأداء: كل مستوى من الوراثة يضيف طبقة من استدعاءات الدوال واستخدام الذاكرة. هذا يمكن أن يؤدي إلى زيادة الحمل على الأداء، خاصة إذا كانت الدوال تُستدعى بشكل متكرر عبر سلسلة الوراثة.**
3. **التشابك الوثيق: شجرة الوراثة العميقة يمكن أن تؤدي إلى تشابك وثيق بين الفئات. التغييرات في الفئة الأساسية يمكن أن تؤثر على جميع الفئات المشتقة، مما يجعل النظام أكثر هشاشة وصعوبة في التعديل دون إدخال أخطاء.**
4. **تقليل المرونة: مع الوراثة العميقة، يصبح من الصعب إجراء تغييرات على التسلسل الهرمي للفئات. إضافة ميزات جديدة أو تعديل الميزات الحالية يمكن أن يكون تحديًا حيث قد يتطلب تغييرات عبر عدة فئات.**
5. **صعوبة في الاختبار: اختبار الفئات في تسلسل وراثة عميق يمكن أن يكون أكثر صعوبة. قد تحتاج اختبارات الوحدة إلى مراعاة سلوك عدة فئات أساسية، مما يجعل من الصعب عزل واختبار المكونات الفردية.**
6. **احتمالية تكرار الشيفرة: في بعض الحالات، يمكن أن تؤدي شجرة الوراثة العميقة إلى تكرار الشيفرة إذا كانت الوظائف المتشابهة تحتاج إلى تنفيذها في فروع متعددة من التسلسل الهرمي.**

================================================================

**Q3:**What are some potential efficiency disadvantages of having very shallow inheritance trees, that is, a large set of classes, A, B, C, and so on, such that all of these classes extend a single class, Z?

**وجود شجرة وراثة سطحية جدًا، حيث تمتد مجموعة كبيرة من الفئات جميعها من فئة واحدة، يمكن أن يؤدي إلى عدة عيوب من حيث الكفاءة:**

1. **زيادة الحمل على الفئة الأساسية: الفئة الأساسية (Z) قد تصبح محملة بالكثير من المسؤوليات، مما يجعلها صعبة الصيانة والفهم. هذا ينتهك مبدأ المسؤولية الواحدة، الذي ينص على أن الفئة يجب أن يكون لها سبب واحد فقط للتغيير.**
2. **قابلية إعادة الاستخدام المحدودة: عندما تمتد جميع الفئات من فئة أساسية واحدة، يمكن أن يحد ذلك من قابلية إعادة استخدام الفئة الأساسية. التغييرات في الفئة الأساسية يمكن أن يكون لها تأثيرات غير مقصودة على جميع الفئات المشتقة، مما يجعل من الصعب إعادة استخدام الفئة الأساسية في سياقات مختلفة.**
3. **قلة المرونة: شجرة الوراثة السطحية يمكن أن تؤدي إلى قلة المرونة في التسلسل الهرمي للفئات. إذا كنت بحاجة إلى إضافة وظائف جديدة أو تعديل الوظائف الحالية، قد تحتاج إلى إجراء تغييرات في الفئة الأساسية، مما يؤثر على جميع الفئات المشتقة.**
4. **زيادة التشابك: جميع الفئات تكون مرتبطة بشكل وثيق بالفئة الأساسية. هذا يعني أن أي تغيير في الفئة الأساسية يمكن أن يكون له تأثير متسلسل على جميع الفئات المشتقة، مما يجعل النظام أكثر هشاشة وصعوبة في التعديل دون إدخال أخطاء.**
5. **صعوبة في الاختبار: اختبار الفئات في شجرة وراثة سطحية يمكن أن يكون تحديًا. نظرًا لأن جميع الفئات تعتمد على الفئة الأساسية، تحتاج إلى التأكد من أن التغييرات في الفئة الأساسية لا تكسر وظائف أي فئة مشتقة. هذا يمكن أن يجعل اختبار الوحدة أكثر تعقيدًا ويستغرق وقتًا أطول.**
6. **احتمالية تكرار الشيفرة: إذا لم توفر الفئة الأساسية جميع الوظائف اللازمة، قد تحتاج الفئات المشتقة إلى تنفيذ وظائف مشابهة، مما يؤدي إلى تكرار الشيفرة.**

**=======================================================**

**Q4**: Draw a class inheritance diagram for the following set of classes: • Class Goat extends Object and adds an instance variable tail and methods milk( ) and jump( ). • Class Pig extends Object and adds an instance variable nose and methods eat(food) and wallow( ). • Class Horse extends Object and adds instance variables height and color, and methods run( ) and jump( ). • Class Racer extends Horse and adds a method race( ). • Class Equestrian extends Horse and adds instance variable weight and isTrained, and methods trot( ) and isTrained( ).

**OBJECT**

**PIG**

**GOAT**

**+TAIL**

**+JUMB()**

**+MILK()**

**HORSE**

**+HEIGHT**

**+COLOR**

**+RUN()**

**+JUMB()**

**NOSE**

**+EAT(FOOD:STRING)**

**+WALLOW()**

سسسسسس

**EQUESTRIAN**

**+WEIGHT**

**+ISTRAINED**

**+TROT()**

**+ISTRAINED()**

RACE

**+RACE()**

**الاسم :امةالملك محمد مظفر**