Data Structure Lab2

Salsabil AL -Warraq

Q1/Q2 : Assume that we change the CreditCard class (see Code Fragment 1.5) so that instance variable balance has private visibility. Why is the following implementation of the PredatoryCreditCard.charge method flawed?

```
public boolean charge(double price) {
boolean isSuccess = super.charge(price);
if (!isSuccess)
charge(5); // the penalty
return isSuccess;
}
```

إن تنفيذ طريقة الشحن في PredatoryCreditCard معيب لعدة أسباب:

الوصول إلى المتغير الخاص: لا يمكن للفئة الفرعية الوصول إلى الرصيد إذا كان خاصًا، مما يمنعها من التحقق مما إذا كانت هناك أموال كافية.

التكرار اللانهائي: إذا فشلت عملية الشراء، فإن استدعاء charge(٥) قد يؤدي إلى حلقة لا نهائية إذا كان الرصيد غير كافٍ.

عيب التصميم: يجب تطبيق العقوبات فقط في ظل ظروف محددة، ولكن عدم القدرة على الوصول إلى الرصيد يجعل هذا مستحيلًا .

Q5: Can two interfaces mutually extend each other? Why or why not?

لا، لا يمكن لواجهتين (interfaces) أن تمدد كل منهما الأخرى بشكل متبادل. السبب في ذلك هو أنه سيؤدي إلى إنشاء علاقة توريث دائرية، وهي غير مسموح بها في Java أو معظم لغات البرمجة الأخرى.

السبب:

- إذا كانت InterfaceA تمدد nterfaceB او InterfaceB تمدد InterfaceA، فسيكون هناك تكرار غير محدود حيث يتطلب كل منهما الآخر ليتم تعريفه أولاً، مما يسبب تناقضًا يؤدي إلى خطأ في الترجمة.(Compilation Error)

Q6: 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.?

1. زيادة التعقيد: مع تزايد عدد المستويات، يصبح فهم الشيفرة وصيانتها أكثر صعوبة.
2. تأثير على الأداء: قد يتطلب استدعاء الدوال في مستويات متعددة من الوراثة وقتًا إضافيًا، مما يؤثر على الأداء.
3. تكرار البيانات: قد يحدث تكرار في البيانات أو السلوكيات عبر الفئات المختلفة، مما يزيد من استهلاك الذاكرة.
4. صعوبة في التعديل: أي تغيير في الفئة الأساسية قد يتطلب تعديلات في جميع الفئات الفرعية، مما يزيد من الجهد المطلوب.
5. تحديات في التخصيص: قد يصبح تخصيص السلوكيات في الفئات الفرعية أكثر تعقيدًا بسبب التداخل في التعريفات.

Q7: 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?

1. نقص التخصص: الفئات الفرعية قد ترث خصائص غير ضرورية، مما يزيد من استهلاك الذاكرة.
2. تكرار الشيفرة: تجاوز طرق الفئة الأساسية يمكن أن يؤدي إلى تكرار الشيفرة، مما يعقد الصيانة.
3. نقطة فشل واحدة: أي خطأ في الفئة الأساسية يؤثر على جميع الفئات الفرعية، مما يزيد من خطر الأخطاء.
4. حمل أداء زائد: الفئة الأساسية قد تحتوي على طرق غير مستخدمة، مما يؤثر على الأداء.
5. صعوبة في الصيانة: تغييرات في الفئة الأساسية تتطلب تعديلات في جميع الفئات الفرعية.
6. تقليل المرونة: صعوبة في تعديل الوظائف دون التأثير على الفئات الفرعية الأخرى.

Q8: Consider the following code fragment, taken from some package: public class Maryland extends State { Maryland( ) { /∗ null constructor ∗/ } public void printMe( ) { System.out.println("Read it."); } public static void main(String[ ] args) { Region east = new State( ); State md = new Maryland( ); Object obj = new Place( ); Place usa = new Region( ); md.printMe( ); east.printMe( ); ((Place) obj).printMe( ); obj = md; ((Maryland) obj).printMe( ); obj = usa; ((Place) obj).printMe( ); usa = md; } } class State extends Region { State( ) { /∗ null ;( )((Place) usa).printMe constructor ∗/ } public void printMe( ) { System.out.println("Ship it."); } } class Region extends Place { Region( ) { /∗ null constructor ∗/ } public void printMe( ) { System.out.println("Box it."); } } class Place extends Object { Place( ) { /∗ null constructor ∗/ } public void printMe( ) { System.out.println("Buy it."); } } What is the output from calling the main( ) method of the Maryland class?

.Read it

.Ship it

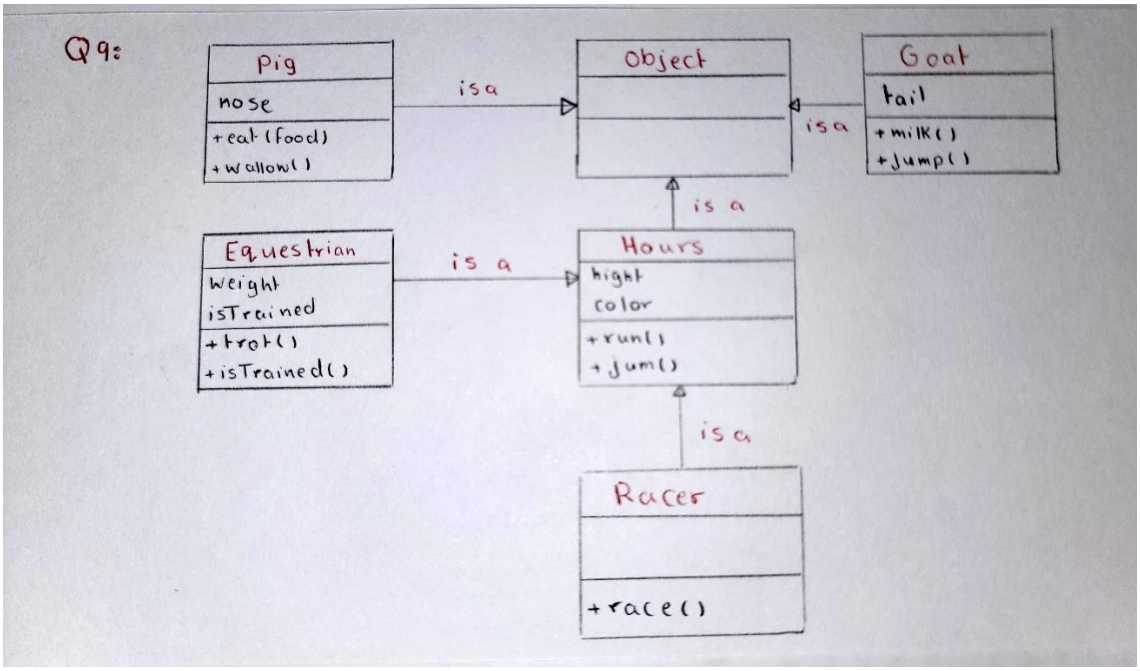
.Buy it

.Read it

.Box it

Read it

Q9: 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



Q10: Consider the inheritance of classes from Exercise R-2.12, and let d be an object variable of type Horse. If d refers to an actual object of type Equestrian, can it be cast to the class Racer? Why or why not?

الإجابة: لا، لا يمكن ذلك.

السبب:

علاقة الوراثة:

Racer و Equestrian كلاهما يرثان من Horse، لكنهما ليسا مرتبطين ببعضهما (لا توجد علاقة وراثة بينهما).

قواعد التحويل:

التحويل يكون ممكنًا فقط إذا كان الكائن الفعلي من النوع المراد التحويل إليه أو نوع فرعي منه.

إذا كان d يشير إلى كائن من نوع Equestrian، فلن يمكن تحويله إلى Racer، وسيحدث خطأ من النوع ClassCastException.

https://github.com/SalsabilMajid/Assignment2.git