1-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;
}
```

The

في حالة نجاح الشحنة الأولي charge(5) المعروض أعلاه غير صحيح بسبب استدعاءالطريقة.

هناك خلل في هذا التنفيذ لعدة أسباب

1. تكرار الشحنة: عندما يتم استدعاء الطريقة charge(5) في حالة نجاح الشحنة الأولي، فإنهيتم شحن مبلغ إضافي قدره ٥. هذا يعني أن العميلسيتم فرض رسوم إضافية على كل شحنة ناجحة،بغض النظر عن قيمتها الفعلية. هذا يمكن أن يؤدي إلى

٢. عدم التحقق من الرصيد: في هذا التنفيذ، لايتم التحقق من الرصيد قبل استدعاء الطريقة charge(5) لفرض العقوبة. قد يؤدي ذلك إلىفرض العقوبة حتى إذا كان الرصيد صفرا أو سالبا، مما

سلوك غير متوقع وغير عادل للعميل.

يزيد من الديون المستحقة على العميل دون التحقق

من قدرته على سدادها.

الحل هذه المشكلة، يجب إجراء التحقق من الرصيد قبلفرض العقوبة وتنفيذها فقط إذا كان الشحن الأصلي ناجاوالرصيد إيجابيا. يمكن تحقيق ذلك عن طريق إضافةشرط

إضافي قبل استدعاء الطريقة charge(5)

2-

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)
super.charge(5); // the penalty
return isSuccess;
}
```

In

العيب في هذا التنفيذ هو أنه يتجاهل الرصيد الفعلي للبطاقة ويضيف فقط رسوم العقوبة بشكل ثابت (٥ وحدات) عندما يفشل الشحن وبالتالي، الحل الصحيح سيكون كالتالي:

```
public boolean charge(double price) {
```

```
boolean isSuccess = super.charge(price);
if (!isSuccess && getBalance() >= 5)
super.charge(5); // العقوبة
return isSuccess;
}
```

في هذا الحل، يتم التحقق من الرصيد المتاح قبل خصم رسوم العقوبة، ويتم إضافة رسوم العقوبة فقط

3-
Give a short fragment of Java code that uses the progression classes from Section 2.2.3 to find the eighth value of a Fibonacci progression that starts with 2 and 2 as its first two values.

```
  public FibonacciProgression() {
    first = 0;
    second = 1;
    prev = second - first;
  }

  protected void advance() {
    long temp = prev;
    prev = current;
    current += temp;
  }

  public static void main(String[] args) {
    FibonacciProgression fibonacci = new FibonacciProgression();
    fibonacci.printProgression(8);
  }
}
```

4 -If we choose an increment of 128, how many calls to the nextValue method from the ArithmeticProgression class of Section 2.2.3 can we make before we cause a long-integer overflow?

إذا تم تجاوز الحد الأقصى للعدد الصحيح الطويل، فسيتم حدوث تجاوز وتغيير في القيمة بشكل غير صحيح، وقد يؤدي ذلك إلى نتائج غير متوقعة أو غير صحيحة في البرنامج

5-Can two interfaces mutually extend each other? Why or why not?

لا، لا يمكن لواجهتين أن تمتدان بشكل متبادل. هذا يعودإلى طبيعة العلاقة بين الواجهات في لغات البرمجة. عندماتمتد واجهة من واجهة أخرى، فإن الواجهة الفرعية)التيتمتد(تحتوي على جميع الأعضاء والتعليمات البرمجيةالمعرفة في الواجهة الأساسية)التي تمتد منها(. وهذا يعنيأنها توفر مزيدا من الوظائف والتعليمات البرمجية.

عندما يحدث تمديد متبادل بين واجهتين، يعني ذلك أنكل منهما يحتوي على أعضاء وتعليمات برمجية من الآخروهذا يؤدي إلى تضارب واضح في التعاريف والوظائفالمتاحة في الواجهتين. وهذا يتعارض مع مفهوم التمديدوالتوسعة الهرمية في البرمجة.

بدلا من ذلك، يمكن للواجهات أن تمتد من واجهات أخربشكل غير متبادل. يمكن للواجهة الفرعية أن تضيف عناصرإضافية إلى الواجهة الأساسية وتوفر بالتالي وظائفإضافية. هذا يسمح بتنظيم وتجميع الوظائف ذات الصلةفي واجهات مختلفة واستخدامها بشكل منفصل أو مجتمعالإنشاء تركيبة مرنة وقابلة للتوسعة من الواجهات.

6-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.?

هناك عدة عيوب في الكفاءة يمكن أن تنشأ عن وجودتسلسلات توريث عميقة جدا، أي مجموعة كبيرة من الفئاتA و B و وهكذا، حيث تمتد B من A، وتمتد من B،وتمتد D من C، وما إلى ذلك. وفيما يلي بعض العيوبالمحتملة لهذا النوع من التوريث العميق

١. زيادة التعقيد وصعوبة الفهم: كلما زادت عمق التسلسل،زادت تعقيدية الشيفرة وصعوبة فهمها. يصعب تتبعالتداخلات والتأثيرات بين الفئات المختلفة، مما يجعل

صيانة الشيفرة وتعديلها أكثر صعوبة..٢ زمن الاستدعاء والأداء: عند استدعاء الأساليب أوالخصائص في سلسلة التوريث العميقة، قد يكون هناكتأثير سلبي على الأداء. يتطلب الوصول إلى الأساليبوالخصائص في كل فئة في السلسلة مروزا بالعديد منالطبقات، وهذا يستهلك وقتا إضافيا ويؤثر على الأداءالعام للتطبيق.

7-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?

تأثير سلبي على الأداء
زيادة في تكرار الكود
زيادة في استهلاك الذاكرة

8-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; ((Place) usa).printMe( ); } } class State extends Region { State( ) { /* null 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

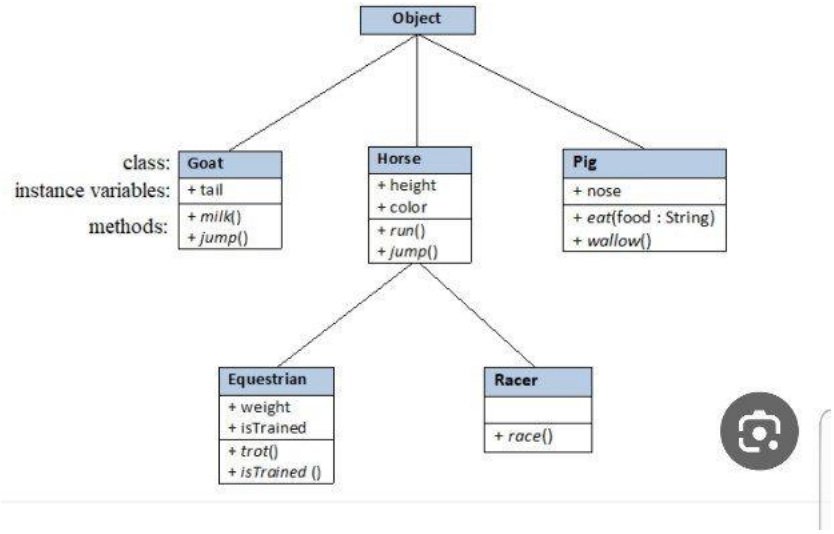
.Box it

.Buy it

.Read it

.Buy it

9-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( ).



10-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?

The

عند استخدام تحويل النوع (casting)، يجب أن يكون النوع المحول إليه نوعًا فرعيًا مباشرًا للنوع الأصلي. في هذه الحالة، فئة Racer ليست فرعية مباشرة لفئة Equestrian، بل هي فرعية مباشرة لفئة Horse. وبالتالي، لا يمكن تحويل المتغير d من نوع Equestrian إلى فئة Racer.

11-Give an example of a Java code fragment that performs an array reference that is possibly out of bounds, and if it is out of bounds, the program catches that exception and prints the following error message: “Don’t try buffer overflow attacks in Java!”

```
public void makePayment(double amount) {
   if (amount < 0) {
      throw new IllegalArgumentException("Payment amount cannot be negative");
   }

   balance -= amount;
}
```

If the parameter to the makePayment method of the CreditCard class (see Code Fragment 1.5) were a negative number, that would have the effect of raising the balance on the account. Revise the implementation so that it throws an IllegalArgumentException if a negative amount is sent as a parameter.

Data Structure Lab2 -Object-Oriented Design

```
public void makePayment(double amount) { // make a
payment if(amount<0) throw new
IllegalArgumentException("Negative Amount is not
Allowed"); balance -= amount; }
```

```
java
public class CreditCard {
   private double balance;

   public void makePayment(double amount) {
      if (amount < 0) {
         throw new IllegalArgumentException("Amount cannot be negative");
      }
      balance += amount;
   }
}
```