**Data Structure Lab2**

***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?***

**Protected double balance;**

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

**boolean kim = super.charge(price);**

**if (!kim){**

**charge(5);**

**return kim;**

**}**

**هنا شوف يكون هنالك خطا لانه سوف يتم استدعاء الى مالانهاية**

**Super سوف يتم استخدام الدالة الأساسية ليتم طباعته مرة واحد**

**Privet double balance;**

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

**boolean kim = super.charge(price);**

**if (!kim)**

**super.charge(5);**

**return kim;**

**}**

***2- Assume that we change the CreditCard class (see Code Fragment 1.5) so that instance variable balance has private visibility.***

***Whyisthefollowingimplementationofthe PredatoryCreditCard.charge method flawed?***

public boolean charge(double price) {

boolean isSuccess = super.charge(price);

if (!isSuccess) {

super.charge(5); // the penalty}

return isSuccess;

}

public boolean penalty(double penaltyAmount) {

return charge(penaltyAmount);

public boolean charge(double price) {

boolean isSuccess = super.charge(price);

if (!isSuccess) {

boolean pSuccess= super. penalty (5);

isSuccess= pSuccess;

}

***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 class FibonacciProgression extends Progression {

protected long prev;

public FibonacciProgression() {

this(0, 1);

}

public FibonacciProgression(long first, long second) {

super(first);

prev = second - first;

}

@Override

protected void advance() {

long temp = prev;

prev = current;

current += temp;

}

}

public class Main {

public static void main(String[] args) {

FibonacciProgression kim= new FibonacciProgression(2, 2);

for (int i = 1; i < 8; i++) {

kim.nextValue();

}

System.out.println("The eighth value of the Fibonacci progression is: " +

kim.nextValue());

}

}

***-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?***

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

**لا، لا يمكن لواجهتين في**

**Java أن تقوم كل منهما بتمديد الأخرى بشكل متبادل.**  
هذا لأن Java لا تسمح بالوراثة الدائرية (Circular Inheritance)، سواء بين الصفوف (Classes) أو الواجهات (Interfaces). إذا حاولت إنشاء واجهتين تقومان بتمديد بعضهما البعض بشكل متبادل، فسيؤدي ذلك إلى خطأ في وقت الترجمة (Compilation Error).

***-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.?***

زيادة وقت البحث عن الطريقة

زيادة الاقتران بين الفئات

صعوبة فهم التعليمات البرمجية وصيانتها

زيادة خطر مشكلة الفئة الأساسية الهشة

السلوك الزائد أو المفرط

التحديات المتعلقة بتعدد الأشكال

تعقيد الاختبار

خطر إساءة استخدام التسلسل الهرمي

***-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?***

**.1زيادة العبء على الفئة الأساسية ( (Z**

**2. صعوبة التخصيص للفئات المشتقة**

**3. زيادة التعقيد في الفئة الأساسية**

**4. انخفاض إمكانية إعادة استخدام الكود**

**5. التحديات في قابلية التوسع في المستقبل**

**6. انتهاك مبدأ المفتوح/المغلق**

***-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.**

**Ship it.**

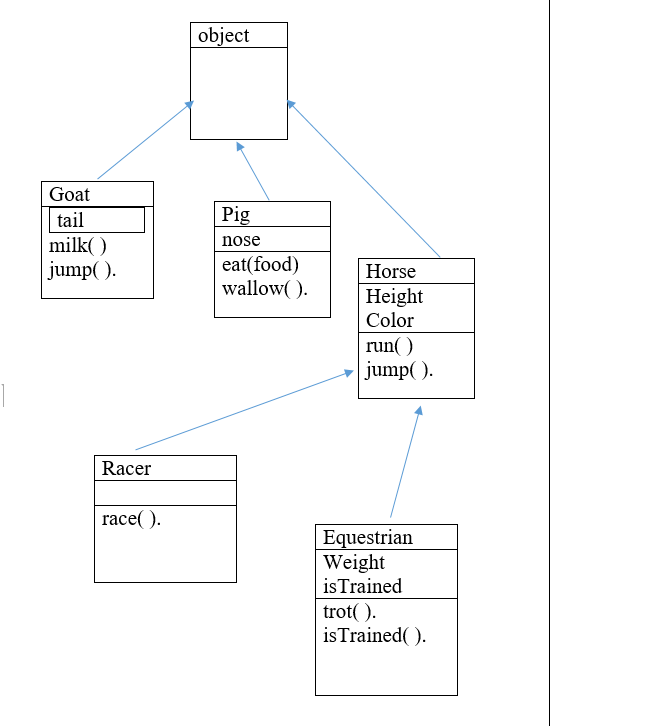
**Buy it.**

**Read it.**

**Box it.**

**Read 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( ).***



* ***-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?***
* ***لا لأنه لا يوجد صله مباشرة بين الكلاسين فهما وارثان من كلاسات الاب لا غير***
* ***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!”***

package labr2.pkg14;

public class ArrayBoundsExample {

public static void main(String[] args) {

int[] numbers = {10, 20, 30, 40, 50};

try {

System.out.println("Attempting to access an out-of-bounds index...");

int value = numbers[10];

System.out.println("Value: " + value);

}

catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Don’t try buffer overflow attacks in Java!");

}

}

}

***12-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.***

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