Exercises and Homework

|  |  |  |
| --- | --- | --- |
| 1 | R-2.4 | 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;  }  سيكون هناك استثناء من نوع overflow في حالة كان price + balance >limit صحيح  اما لوكان غير صحيح سيتمكن المتغير من نوع sub من الوصول الى متغير balence وتغييرة والذي من المفترض ان يكون private  The PredatoryCreditCard.charge method is flawed because it can potentially result in an infinite loop. The method first attempts to charge the specified price using the superclass's charge method. If this attempt fails, the method recursively calls itself, passing a penalty amount of 5. This means that if the initial charge fails, the method will continuously call itself, adding a penalty of 5 to the amount being charged each time. This could eventually lead to a situation where the attempted charge exceeds the credit limit of the account, but the method will continue to recurse indefinitely |
| 2 | R-2.5 | 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;  }  حالة كان price + balance >limit غير صحيح سيتمكن المتغير من نوع sub من الوصول الى متغير balence وتغييرة والذي من المفترض ان يكون private  In either case, you can't be charged a fee if you are close enough to the balance that the fee (of value 5) would exceed your limit. |
| 3 | R-2.6 | 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.  FibonacciProgression fibonacci= new FibonacciProgression(2,2); fibonacci.printProgression(8);  package com.mycompany.slide3\_progression;  public class Fibonacci extends Slide3\_progression{  protected long prev;  public Fibonacci(){this(0,1);}  public Fibonacci(long start,long second){  super(start);  this.prev=second-start;  }  public void advance(){  long temp=prev;  prev=current;  current+=temp;  }    public static void main(String[] args) {  System.out.println("Hello mimchan");  Fibonacci opj=new Fibonacci(2,2);  opj.printProgression(8);    }  } |
| 4 | R-2.7 | 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?  A long-integer overflow occurs when the value of a long variable exceeds the maximum representable value, which is 2^63 - 1 (approximately 9.223 x 10^18). The ArithmeticProgression class generates a sequence of values based on the formula:  يعتمد عدد مرات الاستدعاء على طول السلسة التي تريد  value(n) = first + (n - 1) \* increment  where n is the position of the value in the progression, first is the initial value, and increment is the common difference between consecutive values.  Assuming first is a relatively small positive integer, we can approximate the maximum value of n as:  n ≈ (2^63 - 1) / 128 ≈ 7.18 x 10^12  Therefore, we can make approximately 7.18 x 10^12 calls to the nextValue() method before causing a long-integer overflow. |
| 5 | R-2.8 | Can two interfaces mutually extend each other? Why or why not?  في حال كان المقصود وراثة متبادلة فذلك لايصح لان المترجم لن يعرف من يجب تنفيذه أولا  اما اذا كان المقصود وراثة احدوها من الاخر فذلك صحيح للان هناك دوال وتغيرات ليس عليه إعادة كتابتها  Two interfaces cannot mutually extend each other directly due to the potential for ambiguity and conflicts. Instead, interfaces can be used in conjunction with multiple inheritance to provide the desired functionality without introducing these issues  Cause Cyclic inheritance |
| 6 | R-2.9 | 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.?  من اهم العيوب صعوبة الصيانة والتعديل في الفصول العلياء بسبب الاعتمادية الكبيرة عليها من قبل الفصول السفلى أي التي ترث منها أحيانا قد تكون وراثة غير ضرورية وبالتالي اهدار للمساحة |
| 7 | R-2.10 | 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?  من وجهة نظري واكثر مشكله واجهتني هي صعوبة التحويل أي  Implicit explicit  وذلك عند تحويل أي وريث الى وارث ومن ثم استخدام التحويل من الوارث الى وريث ليس منس نوع الوريث السابق هذا مربك قليلا |
| 8 | R-2.11 | 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.  Bye it.  Read it.  Box it.  Read it; |
| 9 | R-2.12 | 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 | R-2.13 | 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 answer is no because Racer is not sub or super for Equesrain Equestrian cannot be cast to class R\_2\_13.Racer (R\_2\_13.Equestrian and R\_2\_13.Racer are in unnamed module of loader 'app')*  لايصح وذلك لانه لاتوجد بينهم أي علاقة وراثة |
| 11 | R-2.14 | 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 com.mycompany.lab2;  import java.util.Scanner;  public class trayCatch {  public static void main(String[] args) {  System.out.println("Hello World!");  int num[]={1,2,3,4,5};  Scanner s=new Scanner(System.in);  int num1=s.nextInt();  try{  System.out.println(num[num1]);  }  catch(ArrayIndexOutOfBoundsException e){  System.out.println("Don't try buffer overflow attacks in gava");  }  }  }  public static void main(String[] args) {  int[] x = {11, 12, 13, 14, 15};  System.*out*.println("input index to print negative number to exit");  Scanner input = new Scanner(System.*in*);  int y=input.nextInt();  while (y>=0) {  try {  System.*out*.println(x[y]);  } catch (ArrayIndexOutOfBoundsException e |
| 12 | R-2.15 | 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 class throwExer {  private int balance;  public void makePayment(double amount) {  if(amount<0)  throw new IllegalArgumentException("Negative Amount is not Allowed");  balance-= amount;  }  } |