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;  }  قد يدخل الكود في حلقة لا متناهية من الاستدعاءات الذاتية في حالة فشل عملية الشراء وكانت الرسوم الإضافية مع الرصيد الحالي أكبر من الحد الائتماني  بدلا من الكود المعيب هذا يمكن كتابة هذا الكود لإضافة رسوم على العميل في حال حاول الشراء برسوم أكثر من الحد الائتماني 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;  }  هذه الدالة تعيد القيمة المنطقية السابقة ولا تقوم بتحديث قيمة المتغير isSuccess بعد عملية فرض الرسوم وبالتالي قد تعيد قيمة منطقية غير صحيحة  بدلا من الكود المعيب هذا يمكن كتابة هذا الكود لإضافة رسوم على العميل في حال حاول الشراء برسوم أكثر من الحد الائتماني  public boolean charge(double price) {  boolean isSuccess = super.charge(price);  if (!isSuccess) {  // محاولة فرض الرسوم الإضافية مرة واحدة فقط  isSuccess = super.charge(5);  }  return isSuccess;  } |
| 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.  public void printProgression(int n) {  long lastValue;  lastValue = nextValue(); // print first value without leading space  for (int j = 1; j < n; j++) {  lastValue = nextValue(); // print leading space before others  }  System.out.println(lastValue); // print first value without leading space  }  FibonacciProgression fibonacci= new FibonacciProgression(2,2); fibonacci.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?  لحساب ذلك رياضيا يمكننا قسمة اعلى قيمة يمكن تخزينها في متغير long وهي  -1263 على متغير الزيادة increament فيكون الناتج هو 720575940379279936 لم نستطع تنفيذ ذلك بكود برمجي لأن جهاز اللاب خاصتي لم يستطيع القيام بكل هذه الاستدعاءات  in the progression, first is |
| 5 | R-2.8 | Can two interfaces mutually extend each other? Why or why not?  لا يمكن ذلك لأنه يؤدي الى cyclic dependency وهو غير مسموع به في نظام type system لأن ال compiler لا يستطيع فهم ايهم سيعتمد على الاخر مما يؤدي الى الأخطاء |
| 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.?   **زيادة وقت التنفيذ:**   * **استدعاء :constuctors** عند إنشاء كائن لفئة في نهاية السلسلة، يتم استدعاء **constuctors**  جميع الclasses التي ترث منها، مما يزيد من وقت التنفيذ. * **البحث عن الmethods** عندما يحاول المترجم تحديد الطريقة المناسبة لاستدعائها، فإنه يضطر إلى البحث في جميع الفئات في التسلسل الورثي، مما يزيد من وقت التنفيذ.    **زيادة تعقيد الكود:**   * يصعب فهم العلاقات بين الفئات في تسلسل طويل، مما يجعل صيانة وتعديل الكود أكثر صعوبة. * يزيد احتمال حدوث أخطاء غير متوقعة نتيجة للتغييرات التي تطرأ على الفئات العليا في التسلسل.    **ارتباط شديد بين الفئات:**   * يرتبط كل فئة في التسلسل ارتباطًا وثيقًا بالفئات التي ترث منها، مما يعني أن أي تغيير في فئة واحدة قد يؤثر على جميع الفئات الأخرى في التسلسل.    **صعوبة في الاختبار:**   * يصعب اختبار الفئات في تسلسل وراثة عميق بشكل منفصل، حيث أن سلوك كل فئة يعتمد على سلوك الفئات التي ترث منها. |
| 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?   **Increased Method Resolution Overhead:**   * عندما يحتوي العديد من الفئات الفرعية على طرق تحمل نفس الاسم أو أسماء مشابهة، فإن المترجم يضطر إلى البحث بشكل مكثف لتحديد الطريقة الصحيحة التي يجب تنفيذها عند استدعاء هذه الطريقة. هذا يؤدي إلى زيادة وقت التنفيذ.   ** Large Class Hierarchy:**   * إن الحفاظ على فهم عدد كبير من الفئات الفرعية التي ترث مباشرة من فئة أساسية واحدة يمكن أن يكون معقدًا ومعرضًا للأخطاء. * أي تغيير في الفئة الأساسية سيؤثر على جميع الفئات الفرعية، مما يزيد من خطر حدوث عواقب غير مقصودة ويجعل صيانة الكود أكثر صعوبة.    **Potential for Code Bloat:**   * إذا كانت العديد من الفئات الفرعية تشترك في وظائف مشتركة، فقد يؤدي ذلك إلى تكرار الكود وانتفاخ الفئة الأساسية. هذا يجعل الفئة الأساسية كبيرة ومعقدة للغاية.   ** Limited Flexibility:**   * يمكن أن يحد التسلسل الوراثي الضحل من مرونة تصميم الفئات. إذا ظهرت متطلبات جديدة تتطلب دمج ميزات من عدة فئات فرعية، فقد يكون من الصعب تحقيق ذلك دون إدخال مستويات وراثة إضافية أو حلول معقدة. |
| 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  Buy it  Read it  Box it  Buy 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?  لا يمكن الوراثة بين كلاسين ان لم يكن بينهما علاقة وراثة أي ان احدهم ليس أبا للآخر |
| 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!”  public class Lab2 {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  int x []={3,5,6,8};  Scanner in=new Scanner(System.in);  int i = in.nextInt();  try{  System.out.println(x[i]);  }  catch(Exception e){System.out.println("invalid index");}  }    } |
| 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 void makePayment(double amount) { *// make a payment* if(amount<0)  throw new IllegalArgumentException("Negative Amount is not Allowed");  balance -= amount;  } |