1

Exercises and Homework

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;
 }
```

بطريقة معيبة لأنها يمكن أن تؤدي إلى حدوث حلقة تكرارية لا نهائية. تحاول الطريقة أولاً شحن السعر المحدد باستخدام طريقة `PredatoryCreditCard` في `charge` تنفذ الطريقة ، إذا فشلت هذه المحاولة، فإن الطريقة تقوم بالتكرار بنفسها بشكل متكرر، مما يؤدي إلى إضافة غرامة قدرها 5 إلى المبلغ المشحون في كل مرة. `super.charge` الشحن في الفئة الأساسية .هذا يمكن أن يؤدي في نهاية المطاف إلى وجود موقف يتجاوز فيه المبلغ المحاولة الحد الائتماني للحساب، ولكن الطريقة لا تتوقف وتستمر في التكرار بشكل لا نهائي

بمعنى آخر، إذا فشلت محاولة الشحن الأولى بسبب عدم كفاية الرصيد، فإنها ستقوم بشحن المبلغ مرة أخرى بالإضافة إلى غرامة 5، وإذا فشلت هذه المحاولة أيضًا، فإنها ستقوم بتكرار العملية .مرة أخرى ومرة أخرى وهكذا دون توقف. هذا يمكن أن يستمر حتى يتم تجاوز الحد الائتماني والمبلغ المطلوب للشحن، ولكن الطريقة لن تتوقف وستستمر في التكرار بشكل لا نهائي

لحل هذه المشكلة، يجب أن يتم تضمين آلية للتحقق

آخر لتتبع الغرامات المفروضة ومقارنتها بالرصيد المتبقي قبل إجراء المزيد من المحاولات للشحن (private) من الرصيد المتبقي قبل تكرار الشحن.

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;
}

2

:معيبة بسبب عدة أسباب `PredatoryCreditCard` المعروضة في `charge` تعتبر طريقة

1. لشحن المبلغ المحدد. ومع ذلك، إذا فشلت هذه المحاولة، يتم استدعاء طريقة `super. Charge(price)` تغيير الرصيد بشكل مباشر: في هذه الطريقة، يتم استدعاء طريقة الشحن في الفئة الأساسية .super` الشحن في الفئة الأساسية مرة أخرى بإضافة غرامة 5 إلى المبلغ المحاول للشحن. ومع ذلك، لا يتم تحديث الرصيد المحدث بالغرامة المفروضة. هذا يعني أن الرصيد الذي يتم استرجاعه بواسطة .لا يتطابق مع الرصيد الفعلي للبطاقة. وبالتالي، يتم تجاوز القيمة الفعلية للرصيد ويتعذر على المستخدم الحصول على قيمة الرصيد الصحيحة من البطاقة `Charge(price)

2. بغض النظر عن نجاح أو فشل طريقة الشحن المكملة. بمعنى آخر، إذا `super. Charge(price)` التي تم الحصول عليها من `is Success` عدم إعادة القيمة الصحيحة: تقوم الطريقة بإعادة قيمة فشلت محاولة الشحن الأولى، فإن الطريقة ستقوم بشحن

بغض النظر عن نجاح أو فشل هذه المحاولة الثانية. هذا يمكن أن يؤدي إلى تقديم معلومات غير صحيحة للمستخدم حول نجاح أو `true` المبلغ مرة أخرى بالإضافة إلى الغرامة، ومع ذلك، ستعود القيمة .فشل عملية الشحن الفعلية

:لحل هذه المشكلة، يجب إجراء التحديثات التالية في الطريقة

```
public Boolean charge (double price) {
Boolean is Success = super. Charge(price);
if (! is Success)
is Success = super. Charge (price + 5); // Apply penalty
return is Success;
}
```

تعكس نجاح أو فشل عملية الشحن الفعلية بما في ذلك `is Success` .هذه الطريقة تضمن أن القيمة المعادة الغرامة المفروضة

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.

3

من القسم 2.2.3 لإيجاد القيمة الثامنة في تسلسل فيبوناتشي باستخدام الكود التالي `Fibonacci Progression` يمكن استخدام فئة:

```java
Fibonacci Progression progression = new Fibonacci Progression (2, 2);
progression. print Progression (8);
```

print` ونمرر القيمتين الأولى للتسلسل (2 و2) كمعاملات للبناء. ثم نستخدم الطريقة `Fibonacci Progression` في هذا الكود، نقوم بإنشاء كائن من فئة Progression` لطباعة التسلسل حتى القيمة الثامنة.

النتيجة المتوقعة ستكون:

```
2 2 4 6 10 16 26 42
```

بالتالي، القيمة الثامنة في التسلسل هي 42.

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?

4

قبل `next Value` من القسم 2.2.3، فإن عدد المكالمات التي يمكننا القيام بها لطريقة `Arithmetic Progression` قيمتها 128 في فئة (increment) إذا اخترنا زيادة يعتمد على حجم النطاق الممكن للعدد الصحيح الطويل (long integer) للعدد الصحيح الطويل (overflow) حدوث تجاوز.
نظرًا لأن العدد الصحيح الطويل في لغة جافا يستخدم 64 بت لتمثيل الأرقام، فإن النطاق الممكن للعدد الصحيح الطويل يكون بين -9,223,372,036,854,775,808 و9,223,372,036,854,775,807.
ومن ثم طرح 1 لأن القيمة (increment) لحساب عدد المكالمات التي يمكننا القيام بها، يمكننا استخدام النطاق الممكن للعدد الصحيح الطويل مقسومًا على قيمة الزيادة الأولى للتسلسل لا تحتسب كمكالمة. يمكن تنفيذ ذلك في الشفرة التالية:

```java
Arithmetic Progression progression = new
Arithmetic Progression (0, 128);
long Maxville = 9223372036854775807L; // أكبر قيمة ممكنة للعدد الصحيح الطويل
int nuchals = (int) (misvalued / 128) - 1;
```

ونمرر القيمة الأولى (0) وقيمة الزيادة (128) كمعاملات للبناء. ثم نستخدم قيمة العدد الصحيح `Arithmetic Progression` في هذا الكود، نقوم بإنشاء كائن من فئة ونقسمها على قيمة الزيادة (128) ونطرح 1 للحصول على عدد المكالمات التي يمكننا القيام بها قبل حدوث تجاوز العدد الصحيح `Maxville` الطويل الممكنة في المتغير الطويل.
تقريبًا تساوي 7,205,759,222, وهي عدد المكالمات التي يمكننا القيام بها قبل حدوث تجاوز العدد الصحيح الطويل `nuchals` عند تنفيذ الشفرة أعلاه، ستكون قيمة.

5

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

لا يمكن لواجهتين أن تمتدان بعضهما البعض بشكل متبادل. والسبب في ذلك هو أن تمديد الواجهات ينطوي على إنشاء علاقة ترتيبية بينها، حيث تمتد الواجهة الممتدة من الواجهة الممددة. وهذا يتطلب وجود تسلسل واضح للتمديد.

وبناءً على ذلك، فإن تمديد واجهتين بشكل متبادل يعني أن كل B يجب أيضًا أن ينفذ واجهة A ، فإن أي كائن ينفذ واجهةB من واجهة A عندما يتم تمديد واجهة واجهة يجب أن تنفذ الأخرى. وهذا يؤدي إلى حدوث تعارض بين الواجهتين، حيث يتطلب من أي كائن تنفيذهما تنفيذ بعضهما البعض، مما يسبب تضارب في التعاريف ويجعل العلاقة غير متبادلة.

بالتالي، لا يمكن لواجهتين أن تمتدان بعضهما البعض بشكل متبادل

6

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?

D و B تمتد من C و A من B وهكذا حيث تمتد C و B و A هناك بعض العيوب المحتملة من حيث الكفاءة عند وجود أشجار التوريث العميقة جدًا، أي مجموعة كبيرة من الفئات وهكذا C تمتد من.

زيادة في استهلاك الذاكرة: مع كل طبقة جديدة من التمديد، يتم إنشاء نسخة جديدة من الفئة الأصلية مع إضافة السمات والسلوك الجديدة. هذا يؤدي إلى زيادة استهلاك الذاكرة . لأنه يجب تخزين مساحة إضافية لكل فئة في الشجرة.

زيادة في وقت البناء والتهيئة: مع كل فئة في الشجرة، يجب إنشاء وتهيئة الكائنات المرتبطة بها. عند إنشاء كائن في الطبقة الأخيرة من الشجرة، يجب على النظام إنشاء . وتهيئة كافة الأنساق والمراجع العائدة للفئات السابقة في الشجرة، مما يتسبب في زيادة في وقت البناء والتهيئة.

زيادة في تعقيد الصيانة: مع زيادة عدد الفئات في الشجرة، يصبح من الصعب فهم العلاقات وتتبعها. يمكن أن يؤدي ذلك إلى صعوبة في تحديد نقاط الفشل أو .

تنفيذ التغييرات والتحسينات في الشجرة.

ارتفاع تكلفة الاختبار: مع زيادة عدد الفئات في الشجرة، يصبح من الصعب اختبار واختبار جميع السيناريوهات المحتملة والحالات الحدودية في الشجرة. هذا يستلزم مزيدًا من . الموارد والوقت لضمان سلوك صحيح ومتوافق لكافة الفئات.

بشكل عام، يجب أخذ هذه العيوب في الاعتبار عند تصميم تركيبة التوريث. ينبغي تجنب إنشاء أشجار توريث عميقة جدًا إلا إذا كان هناك حاجة ملحة إلى هذا النوع من التصميم وكانت المزايا تفوق العيوب

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?

وهكذا حيث تمتد C و B و A هناك بعض العيوب المحتملة من حيث الكفاءة عند وجود أشجار التوريث الضحلة جدًا، أي مجموعة كبيرة من الفئات جميع هذه الفئات من فئة واحدة.

زيادة في تكرار الكود: في حالة وجود فئات متعددة تمتد من فئة واحدة، قد يكون هناك الكثير من التكرار في الكود. إذا كانت هناك سمات أو سلوك مشترك بين تلك الفئات، فقد يتم تكرار تعريفها في كل فئة، مما يؤدي إلى زيادة في حجم الكود وتكرار الجهود.

صعوبة في إدارة التغييرات: عندما يتم إجراء تغيير في الفئة الأساسية، قد يكون من الصعب تتبع تأثير هذا التغيير على جميع الفئات الفرعية. يجب تحديث كل فئة فرعية يدويًا لمواكبة التغييرات، مما يمكن أن يؤدي إلى أخطاء وقد يستدعي تكرار الجهود في تطبيق التغييرات.

قيود التصميم: عندما تمتد جميع الفئات من فئة واحدة، فإنه يكون هناك قيود تصميمية على الفئات الفرعية. يجب أن تلتزم الفئات الفرعية بالسمات والسلوك المعرفة في الفئة الأساسية، مما يقيد المرونة وقدرة الفئات الفرعية على التكيف مع متطلبات خاصة.

زيادة في استهلاك الذاكرة: مع وجود فئات فرعية تمتد من فئة واحدة، قد يتم تخزين مساحة إضافية في الذاكرة لتخزين معلومات الفئات الفرعية، حتى إن كان لديها سمات وسلوك يتم تمريرها من الفئة الأساسية.

بشكل عام، يجب أخذ هذه العيوب في الاعتبار عند تصميم تركيبة التوريث. ينبغي استخدام التوريث الضحل عندما يكون هناك علاقة وثيقة وقوية بين الفئات الفرعية والفئة الأساسية، وعندما تكون هناك حاجة للحصول على السمات والسلوك المشتركة دون تكرار الكود

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[ ] arg { Region east = new State( ); State md = new Maryland( ); Object obj = new Plac); Place usa = new Region( ); md.printMe( ); east.printMe( ); ((Place) obj).printM); 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?

:، سيتم طباعة النصوص التاليةMaryland في فئة () main عند استدعاء الأسلوب

:تُطبع هذه النصية من الأسطر التالية - "Read it."
imprinted ();
.Maryland في كائن من فئة () print Me هنا، تُستدعى الطريقة

:تُطبع هذه النصية من الأسطر التالية - "Box it."
Eastpointe ();
.Region ، ويتم تعيينه إلى كائن من فئةState في كائن من فئة () print Me هنا، تُستدعى الطريقة

:تُطبع هذه النصية من الأسطر التالية - "Buy it."
((Place) obj).print Me ();
.() print Me ثم يُستدعى الأسلوب Place إلى كائن من فئة obj هنا، يتم تحويل الكائن

:تُطبع هذه النصية من الأسطر التالية - "Read it."
((Maryland) obj).print Me ();
إلى كائن من فئة obj هنا، يتم تحويل الكائن

.() print Me ثم يُستدعى الأسلوب Maryland

:تُطبع هذه النصية من الأسطر التالية - "Box it."
((Place) obj).print Me ();
.() print Me ثم يُستدعى الأسلوب Place إلى كائن من فئة obj هنا، يتم تحويل الكائن

:تُطبع هذه النصية من الأسطر التالية - "Ship it."
((Place) use).print Me ();
.() print Me ثم يُستدعى الأسلوب Place إلى كائن من فئة use هنا، يتم تحويل الكائن

:تُطبع هذه النصية من الأسطر التالية - "Ship it."
((Place) use).print Me ();
.() print Me ثم يُستدعى الأسلوب Place إلى كائن من فئة use هنا، يتم تحويل الكائن

:الإجمالي، ستكون النتيجة كالتالي
Read it.
Box it.
Buy it.

Read it.
Box it.
Ship it.
Ship 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( ).

:تتكون الهيكلة التريثية للفئات المذكورة من الشكل التالي

```
Object
  |
Horse
 /  \
Racer Equestrian
      /  \
   Goat  Pig
```

.Java هي الفئة الأساسية لجميع الفئات في Object الفئة -
.() jump و () run والأساليب color و height وتضيف المتغيرات الوصفية Object تمتد من Horse الفئة -
.() race وتضيف الأسلوب Horse تمتد من Racer الفئة -
وتضيف Horse تمتد من Equestrian الفئة -

.() is Trained و () trot والأساليب is Trained و weight المتغيرات الوصفية
.() jump و () milk والأساليب tail وتضيف المتغير الوصفي Object تمتد من Goat الفئة -
.() wallow و eat(food) والأساليب nose وتضيف المتغير الوصفي Object تمتد من Pig الفئة -

.هذا هو الترتيب الهرمي للتوريث بين الفئات المذكورة

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?

.Horse بنوع d وفي السؤال الحالي، يتم تعيين متغير الكائن .Horse من الفئة Racer ، تمتد الفئةR-2.12 في تمرين

"هي "تمتد من Racerو Equestrian السبب في ذلك هو أن العلاقة بين .Racer ، فلا يمكن تحويله إلى فئةEquestrian تشير إلى كائن فعلي من نوع d إذا كانت .Horse يمثل نوعًا آخر متخصصًا من Equestrian بينما Horse يمثل نوعًا متخصصًا من Racer بمعنى آخر، فإن .(contains) "وليس "تتضمن (extends)

، يتم السماح بتحويل الكائنات إلى الفئات التي تمتد منها (أي الفئات الأكثر متخصصة). ولكن لا يمكن تحويل الكائنات إلى (casting) عند استخدام تحويل النوع .الفئات التي يتم تمديدها منها (أي الفئات الأكثر عامة)

.Racer إلى Equestrian لذلك، في هذه الحالة، لا يمكن تحويل الكائن من نوع

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!”

يقوم بالإشارة إلى مصفوفة خارج حدودها، وفي حالة خروجها عن الحدود، يتم التقاط الاستثناء وطباعة رسالة الخطأ Java **إليك مثالًا لمقطع من الشفرة في التالية: "!لا تحاول هجمات تجاوز الحافة في جافا".**

```java
try {
int[] array = new int[5];
int value = array[10]; // محاولة الإشارة إلى عنصر خارج حدود المصفوفة

// إذا تم الوصول إلى هذا السطر، فإن الإشارة إلى المصفوفة كانت ضمن الحدود
System.out.println(value);
} catch (ArrayIndexOutOfBoundsException e) {
// التقاط الاستثناء عند خروج الإشارة عن حدود المصفوفة
System.out.println("لا تحاول هجمات تجاوز الحافة في جافا!");
}
```

الذي هو خارج حدود المصفوفة. يتم التقاط استثناء `array[10]` بحجم 5، ومن ثم يتم إشارة إلى العنصر `array` في هذا المثال، يتم إنشاء مصفوفة `array` ويتم طباعة رسالة الخطأ المطلوبة "لا تحاول هجمات تجاوز الحافة في جافا!" `ArrayIndexOutOfBoundsException`.

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.

إذا كان المبلغ سالبًا ، فسيؤدي ذلك إلى .CreditCard في فئة ()makePayment ، يتم تمرير المبلغ كمعامل إلى طريقة (Code Fragment 1.5) في النص الأصلي
:عند تمرير مبلغ سالب كمعامل، يمكن استخدام الشفرة التالية IllegalArgumentException زيادة الرصيد في الحساب. لتعديل النص ليتم رفع استثناء

```java
public void makePayment(double amount) {
if (amount < 0) {
throw new IllegalArgumentException("لا يمكن قبول مبلغ سالب.");
}
balance -= amount;
}
```

للتحقق مما إذا كان المبلغ (deduct the payment) في هذا التعديل، يتم إضافة شرط قبل تنفيذ الدفعة

مع رسالة الخطأ المناسبة "لا يمكن قبول مبلغ سالب." سيتم التقاط هذا الاستثناء IllegalArgumentException سالبًا. إذا كان المبلغ سالبًا، يتم رفع استثناء
.makePayment() والتعامل معه في الكود الذي يستدعي طريقة

بهذا التعديل، سيتم رفض المبالغ السالبة ولن يتم زيادة الرصيد على الحساب

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

:لرفض المبالغ السالبة. هنا هي شرح الشفرة باللغة العربية CreditCard في فئة ()makePayment الشفرة المعطاة تقوم بتعديل طريقة

```java
public void makePayment(double amount) {
// إذا كان المبلغ سالبًا، يتم رفع استثناء IllegalArgumentException
if (amount < 0) {
throw new IllegalArgumentException("المبلغ السالب غير مسموح به.");
}
// يتم خصم المبلغ من الرصيد
balance -= amount;
}
```

، إذا كان()makePayment عند استدعاء طريقة

`if` ويتم إرسال رسالة الخطأ "المبلغ السالب غير مسموح به." تمامًا كما هو موضح في الشرط IllegalArgumentException المبلغ الذي يتم تمريره سالبًا، سيتم رفع استثناء
.`(amount < 0)`

بهذا التعديل، سيتم رفض المبالغ السالبة ولن يتم خصمها من الرصيد