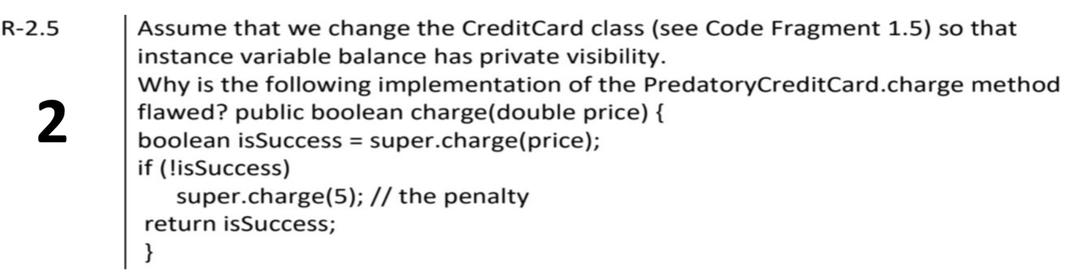


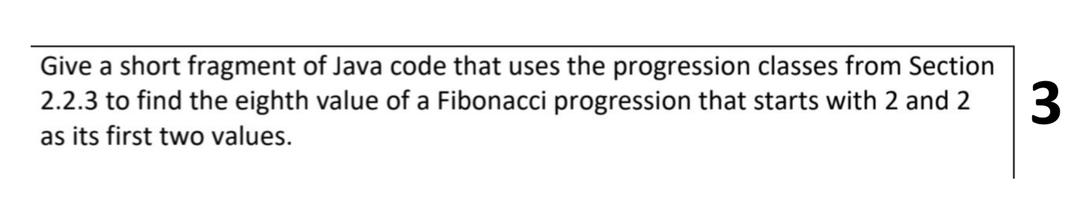
بطريقة معيبة يمكن أن تؤدي إلى حدوث حلقة تكرارية لا نهائية. تحاول الطريقة أوالاً شحن السعر المحدد باستخدام طريقة `PredatoryCreditCard `في `charge `تنفذ الطريقة ، إذا فشلت هذه المحاولة، فإن الطريقة تقوم بالتكرار بنفسها بشكل متكرر، مما يؤدي إلى إضافة غرامة قدرها 5 إلى المبلغ المشحون في كل مرة. `charge.super `الشحن في الفئة الاساسية .هذا يمكن أن يؤدي في نهاية المطاف إلى وجود موقف يتجاوز فيه المبلغ المحاولة الحد الائتماني للحساب، ولكن الطريقة ال تتوقف وتستمر في التكرار بشكل ال نهائي بمعنى آخر، إذا فشلت محاولة الشحن الاولى بسبب عدم كفاية الرصيد، فإنها ستقوم بشحن المبلغ مرة أخرى بالإضافة إلى غرامة ،5 وإذا فشلت هذه المحاولة أيضا، 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 ال 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.



:معيبة بسبب عدة أسباب `PredatoryCreditCard `المعروضة في `charge `تعتبر طريقة لشحن المبلغ المحدد. ومع ذلك، إذا فشلت هذه المحاولة، يتم استدعاء طريقة `(price(Charge .super `تغيير الرصيد بشكل مباشر:

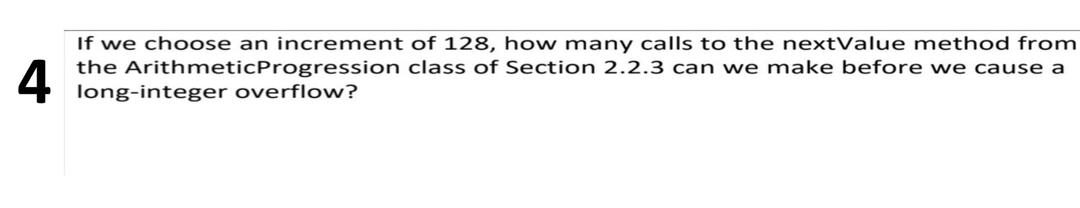
في هذه الطريقة، يتم استدعاء طريقة الشحن في الفئة الأساسية 1. .super `الشحن في الفئة الأساسية مرة أخرى بإضافة غرامة 5 إلى المبلغ المحاول للشحن. ومع ذلك، يتم تحديث الرصيد المحدث بالغرامة المفروضة. هذا يعني أن الرصيد الذي يتم استرجاعه بواسطة .ال يتطابق مع الرصيد الفعلي للبطاقة. وبالتالي، يتم تجاوز القيمة الفعلية للرصيد ويتعذر على المستخدم الحصول على قيمة الرصيد الصحيحة من البطاقة `(price(Charge بغض النظر عن نجاح أو فشل طريقة الشحن المكملة. بمعنى آخر، إذا `(price(Charge .super `التي تم الحصول عليها من `Success is `عدم إعادة القيمة الصحيحة: تقوم الطريقة بإعادة قيمة 2. فشلت محاولة الشحن الاولى، 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 `Success is `.هذه الطريقة تضمن أن القيمة المعادة الغرامة المفروضة.

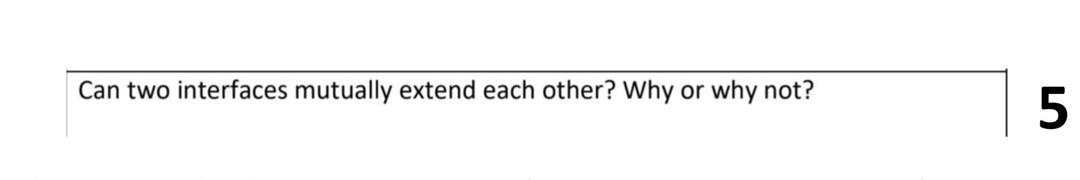


:من القسم 2.2.3 لإيجاد القيمة الثامنة في تسلسل فيبوناتشي باستخدام الكود التالي `Progression Fibonacci `

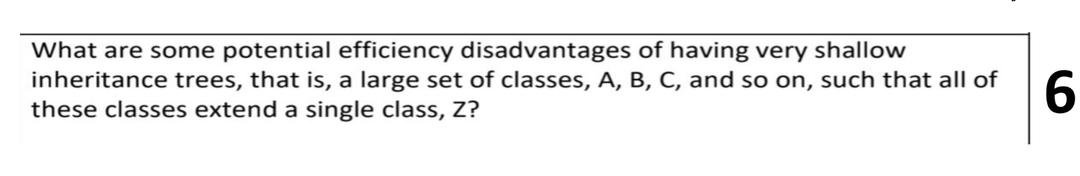
يمكن استخدام فئة ```java Fibonacci Progression progression = new Fibonacci Progression (2, 2); progression. print Progression (8); ``` print `ونمرر القيمتين الاولى للتسلسل ( 2 و2) كمعاملات للبناء. ثم نستخدم الطريقة `Progression Fibonacci `في هذا الكود، نقوم بإنشاء كائن من فئة .لطباعة التسلسل حتى القيمة الثامنة `Progression :النتيجة المتوقعة ستكون ``` 2 2 4 6 10 16 26 42 ``` .بالتالي، القيمة الثامنة في التسلسل هي 42



قبل `Value next `من القسم ،2.2.3 فإن عدد المكالمات التي يمكننا القيام بها لطريقة `Progression Arithmetic `قيمتها 128 في فئة (increment) إذا اخترنا زيادة .يعتمد على حجم النطاق الممكن للعدد الصحيح الطويل (integer long) للعدد الصحيح الطويل (overflow )حدوث تجاوز 64 بت لتمثيل الأرقام، فإن النطاق الممكن للعدد الصحيح الطويل يكون بين 9,223,372,036,854,775,808- ا ألى العدد الصحيح الطويل في لغة جافا يستخدم .و9,223,372,036,854,775,807 ومن ثم طرح 1 ألن القيمة (increment (لحساب عدد المكالمات التي يمكننا القيام بها، يمكننا استخدام النطاق الممكن للعدد الصحيح الطويل مقسوًما على قيمة الزيادة :الأولى للتسلسل تحتسب كمكالمة. يمكن تنفيذ ذلك في الشفرة التالية ```java Arithmetic Progression progression = new Arithmetic Progression (0, 128); أكبر قيمة ممكنة للعدد الصحيح الطويل // ;L9223372036854775807 = Maxville long int nuchals = (int) (misvalued / 128) - 1; ``` ونمرر القيمة الأولى )0( وقيمة الزيادة ) 128( كمعاملات للبناء. ثم نستخدم قيمة العدد الصحيح `Progression Arithmetic `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 أعاله، ستكون قيمة

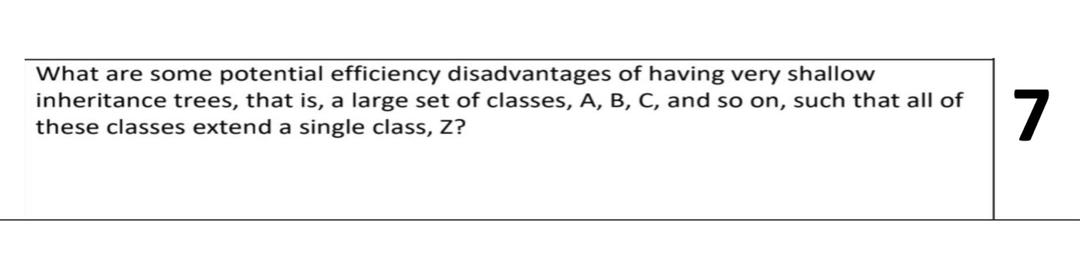


لا يمكن لواجهتين أن تمتدان بعضهما البعض بشكل متبادل. والسبب في ذلك هو أن تمديد الواجهات ينطوي على إنشاء علاقة ترتيبية بينها، حيث تمتد الواجهة الممتدة من الواجهة الممددة. وهذا يتطلب وجود تسلسل واضح للتمديد وبنا ًء على ذلك، فإن تمديد واجهتين بشكل متبادل يعني أن كل .B يجب أيضا أن ينفذ واجهة A ، فإن أي كائن ينفذ واجهة B من واجهة A عندما يتم تمديد واجهة يجب أن تنفذ الاخرى. وهذا يؤدي إلى حدوث تعارض بين الواجهتين، حيث يتطلب من أي كائن تنفيذ بعضهما البعض، مما يسبب تضارب في .التعاريف ويجعل العالقة غير متبادلة بالتالي، ال يمكن لواجهتين أن تمتدان بعضهما البعض بشكل متبادل

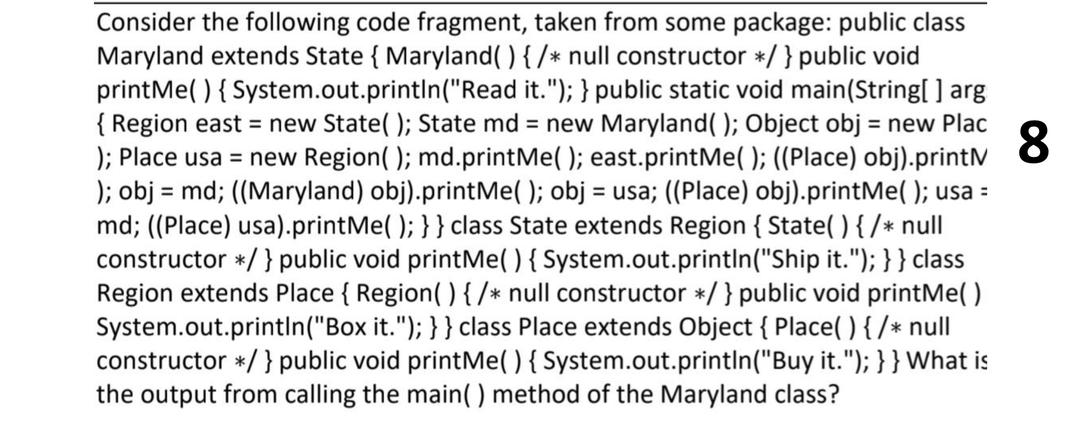


هناك بعض العيوب المحتملة من حيث الكفاءة عند وجود أشجار التوريث العميقة جدا، أي مجموعة كبيرة من الفئات .وهكذا C تمتد من D و B تمتد من C و A من B

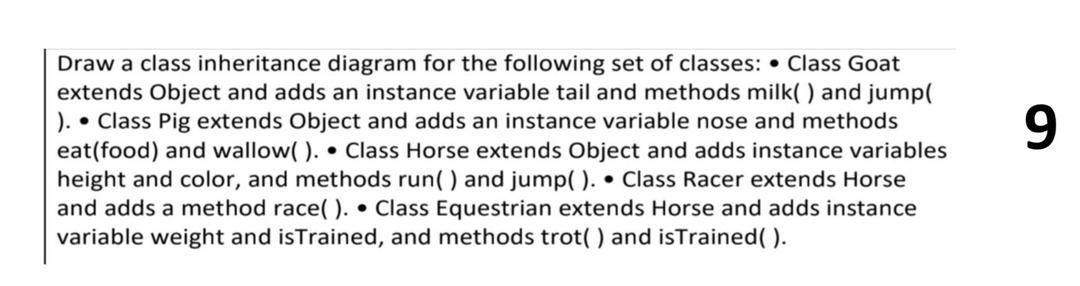
زيادة في استهلاك الذاكرة: مع كل طبقة جديدة من التمديد، يتم إنشاء نسخة جديدة من الفئة الأصلية مع إضافة السمات والسلوك الجديدة. هذا يؤدي إلى زيادة استهلاك الذاكرة . .ألنه يجب تخزين مساحة إضافية لكل فئة في الشجرة زيادة في وقت البناء والتهيئة: مع كل فئة في الشجرة، يجب إنشاء وتهيئة الكائنات المرتبطة بها. عند إنشاء كائن في الطبقة الأخيرة من الشجرة، يجب على النظام إنشاء . .وتهيئة كافة الانسياق 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 العالقات 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 مزيًدا 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 جدا إلى إذا كان هناك حاجة ملحة إلى هذا النوع من التصميم وكانت المزايا تفوق العيوب



وهكذا حيث تمتد C و B و A هناك بعض العيوب المحتملة من حيث الكفاءة عن د وجود أشجار التوريث الضحلة جدا، 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 اس تهالك 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 عالقة وثيقة وقوية بين الفئات الفرعية والفئة الأساسية، وعندما تكون هناك حاجة للحصول على السمات والسل وك المشتركة دون تكرار الكود



سيتم طباعة النصوص التاليةMaryland في فئة () main عند استدعاء الأسلوب :تُطبع هذه النصية من الأسطر التالية - ".it Read" imprinted (); .Maryland في كائن من فئة () Me print هنا، تُستدعى الطريقة :تُطبع هذه النصية من الأسطر التالية - ".it Box" Eastpointe (); .Region ، ويتم تعيينه إلى كائن من فئةState في كائن من فئة () Me print هنا، تُستدعى الطريقة :تُطبع هذه النصية من الأسطر التالية - ".it Buy" ((Place) obj).print Me (); .() Me print ثم يُستدعى الأسلوب Place إلى كائن من فئة obj هنا، يتم تحويل الكائن :تُطبع هذه النصية من الأسطر التالية - ".it Read" ((Maryland) obj).print Me (); إلى كائن من فئة obj هنا، يتم تحويل الكائن .() Me print ثم يُستدعى الأسلوب Maryland :تُطبع هذه النصية من الأسطر التالية - ".it Box" ((Place) obj).print Me (); .() Me print ثم يُستدعى الأسلوب Place إلى كائن من فئة obj هنا، يتم تحويل الكائن :تُطبع هذه النصية من الأسطر التالية - ".it Ship" ((Place) use).print Me (); .() Me print ثم يُستدعى الأسلوب Place إلى كائن من فئة use هنا، يتم تحويل الكائن :تُطبع هذه النصية من الأسطر التالية - ".it Ship" ((Place) use).print Me (); .() Me print 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



تتكون الهيكلة التريثية للفئات المذكورة من الشكل التالي

Object

/

Horse

/ \

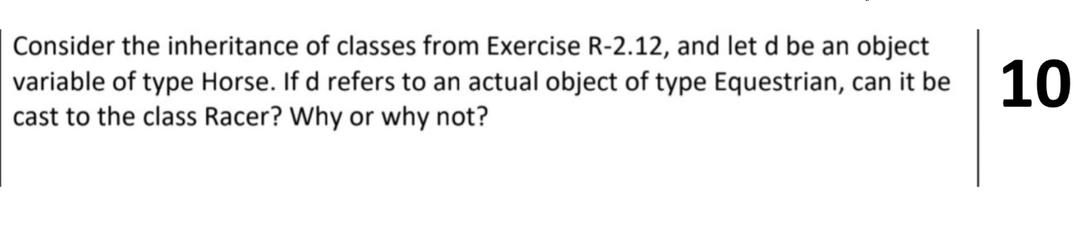
Racer Equestrian

/ \

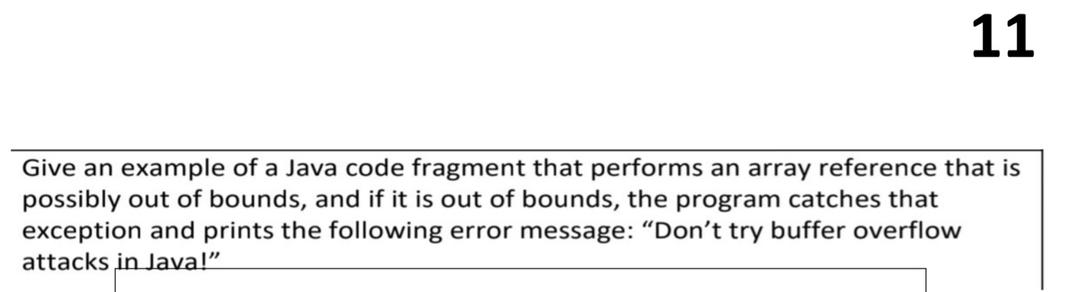
Goat Pig

```

``` .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 - الوصفية المتغيرات weight و is Trained والأساليب trot () و is Trained (). .() jump و () milk والأساليب tail وتضيف المتغير الوصفي Object تمتد من Goat الفئة - .() wallow و (food(eat والأساليب nose وتضيف المتغير الوصفي Object تمتد من Pig الفئة - .هذا هو الترتيب الهرمي للتوريث بين الفئات المذكورة



.Horse بنوع d وفي السؤال الحالي، يتم تعيين متغير الكائن .Horse من الفئة Racer ، تمتد الفئة-2.12R في تمرين "هي "تمتد من Racerو Equestrian السبب في ذلك هو أن العالقة بين .Racer ، فال يمكن تحويله إلى فئةEquestrian تشير إلى كائن فعلي من نوع d إذا كانت .Horse يمثل نوعا آخر متخصصا من Equestrian بينما Horse يمثل نوعا متخصصا 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



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

```java

try {

int[] array = new int[5];

محاولة الإشارة إلى عنصر خارج حدود المصفوفة // ;[10]array = value int

إذا تم الوصول إلى هذا السطر، فإن الإشارة إلى المصفوفة كانت ضمن الحدود //

System.out.println(value);

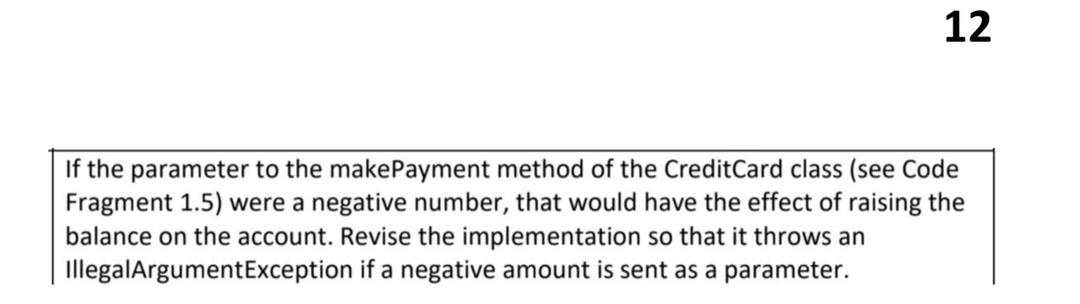
} catch (ArrayIndexOutOfBoundsException e) {

التقاط الاستثناء عند خروج الإشارة عن حدود المصفوفة //

;("!ال تحاول هجمات تجاوز الحافة في جافا")println.out.System

} ```

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



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

```java

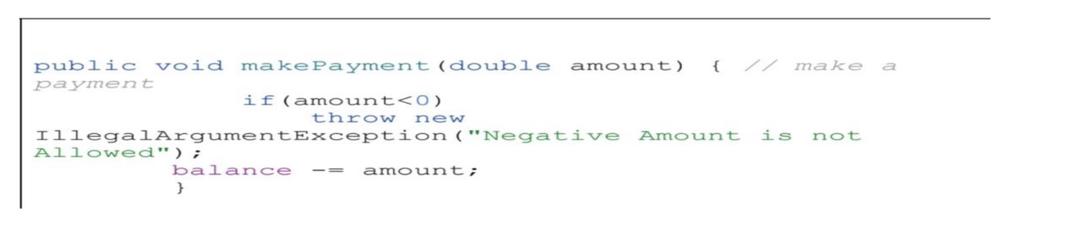
public void makePayment(double amount) {

if (amount < 0) {

{balance -= amount; }

```

للتحقق مما إذا كان المبلغ (payment the deduct (في هذا التعديل، يتم إضافة شرط قبل تنفيذ الدفعة مع رسالة الخطأ المناسبة "ال 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



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

```java

public void makePayment(double amount) {

IllegalArgumentException إذا كان المبلغ سالبًا، يتم رفع استثناء //

if (amount < 0) {

;(".المبلغ السالب غير مسموح به")IllegalArgumentException new t

} يتم خصم المبلغ من الرصيد // balance -= amount; }

```

إذا كان()makePayment عند استدعاء طريقة if `ويتم إرسال رسالة الخطأ "المبلغ السالب غير مسموح به." تما ًما 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