Q10: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، لا يمكن تحويل الكائن `d` من النوع `Equestrian` إلى النوع `Racer`. السبب في ذلك هو أن `Equestrian` و`Racer` هما فئتان شقيقتان (sibling classes) في شجرة الوراثة، حيث أن كلاهما يمتد من الفئة `Horse`.

في Java ، يمكن تحويل الكائنات فقط إلى الفئات التي هي جزء من تسلسل الوراثة الخاص بها. بمعنى آخر، يمكن تحويل الكائن إلى فئته الأساسية أو إلى أي فئة يمتد منها بشكل مباشر أو غير مباشر. ولكن لا يمكن تحويل الكائن إلى فئة شقيقة لأنها ليست جزءًا من تسلسل الوراثة الخاص به.

==========

Q1 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;
}
```

الحل

The implementation of the `PredatoryCreditCard.charge` method is flawed for the following reasons:

1. **Access to Private Variable: If the `balance` variable in the `CreditCard` class is private, the `PredatoryCreditCard` class cannot directly access or modify it. The `super.charge(price)` call will work if the `charge` method in the `CreditCard` class is public or protected, but any direct access to `balance` within `PredatoryCreditCard` would not be allowed.**

2. **Recursive Call: The line `charge(5);` within the `if (!isSuccess)` block is problematic because it calls the `charge` method of `PredatoryCreditCard` again. If the initial charge fails (e.g., due to insufficient credit), the penalty charge of 5 will also likely fail, leading to another recursive call. This can result in an infinite loop and eventually a `StackOverflowError`.**
3. **Penalty Application: The penalty should be applied in a way that does not cause recursive calls. Instead, it should be handled separately to ensure that it does not trigger another charge attempt.**

الكود

```
} public class PredatoryCreditCard extends CreditCard

؛private double penalty = 5.0


public PredatoryCreditCard(String cust, String bk, String acnt,
                } double lim, double initialBal)

؛super(cust, bk, acnt, lim, initialBal)

{


Override@

} public boolean charge(double price)

؛boolean isSuccess = super.charge(price)

} if (!isSuccess)

Apply penalty separately //

؛super.charge(penalty)

{

؛return isSuccess

{
```

Q2: 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;
}
```

الحل

تغيير رؤية المتغير `balance` إلى خاصة (private) في فئة `CreditCard` يؤثر على كيفية تعامل الفئات الفرعية مع هذا المتغير. دعنا نحلل لماذا تكون الطريقة `charge` في فئة `PredatoryCreditCard` معيبة:

١. الوصول إلى المتغير الخاص: إذا كان المتغير `balance` خاصًا في فئة `CreditCard`، فإن الفئة الفرعية `PredatoryCreditCard` لا يمكنها الوصول إليه مباشرة. الطريقة `super.charge(price)` ستعمل إذا كانت الطريقة `charge` في فئة `CreditCard` عامة أو محمية، ولكن أي وصول مباشر إلى `balance` داخل `PredatoryCreditCard` لن يكون مسموحًا.

٢. استدعاء متكرر: السطر `super.charge(5);` داخل الكتلة `if (!isSuccess)` يمثل مشكلة لأنه يستدعي الطريقة `charge` في `PredatoryCreditCard` مرة أخرى. إذا فشل الشحن الأولي (على سبيل المثال، بسبب عدم كفاية الرصيد)، فإن شحن العقوبة بقيمة ٥ سيؤدي أيضًا إلى الفشل على الأرجح، مما يؤدي إلى استدعاء متكرر. هذا يمكن أن يؤدي إلى حلقة لا نهائية وفي النهاية إلى خطأ `StackOverflowError`.

٣. تطبيق العقوبة: يجب تطبيق العقوبة بطريقة لا تسبب استدعاءات متكررة. بدلاً من ذلك، يجب التعامل معها بشكل منفصل لضمان أنها لا تؤدي إلى محاولة شحن أخرى.

الكود للتوضيح

```
} public class PredatoryCreditCard extends CreditCard
```

```
;private double penalty = 5.0


public PredatoryCreditCard(String cust, String bk, String
} acnt, double lim, double initialBal)
;super(cust, bk, acnt, lim, initialBal)
{


Override@
} public boolean charge(double price)
;boolean isSuccess = super.charge(price)
} if (!isSuccess)
Apply penalty separately //
;super.charge(penalty)
{
;return isSuccess
{
  {
```

----==========---------========----------=======

Q3: 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.

```
;package com.mycompany.q3
} public class FibonacciProgression
```

```
protected long current;
protected long prev;

public FibonacciProgression() {
    this(0, 1);
}

public FibonacciProgression(long first, long second) {
    current = first;
    prev = second - first;
}

public long nextValue() {
    long temp = prev;
    prev = current;
    current += temp;
    return current;
}

public static void main(String[] args) {
    FibonacciProgression prog = new FibonacciProgression(2, 2);
    for (int i = 1; i < 8; i++)
        prog.nextValue();
```

```
                                                                {
        System.out.println("The eighth value is: " +
                                                        ؛prog.nextValue())
                                                                {
                                                                {
```

Q4: 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?

```
؛package com.mycompany.q4

} public class Progression

instance variable //

؛protected long current


/* .Constructs a progression starting at zero **/

} ()public Progression

؛(٠)this

{


/* .Constructs a progression with given start value **/

} public Progression(long start)

؛current = start

{


/* .Returns the next value of the progression **/
```

```
} ()public long nextValue

;long answer = current

advance(); // this protected call is responsible for
                    advancing the current value

;return answer

{


Advances the current value to the next value of the  **/
                                    /* .progression

} ()protected void advance

;++current

{


Prints the next n values of the progression, separated  **/
                                    /* .by spaces

} public void printProgression(int n)

System.out.print(nextValue()); // print first value
                    without leading space

} for (int j = 1; j < n; j++)

System.out.print(" " + nextValue()); // print leading
                    space before others

{

System.out.println(); // end the line

{
```

```
} public static void main(String[] args)
'()Progression prog = new Progression
prog.printProgression(10); // Print the first 10 values of
                                the progression
                                {
                                {
```

Q8: 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?

```
package com.mycompany.q8;
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 Region();
      Place usa = new Place();
      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."); }
}
```

Q11: 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.q11;
public class ArrayExample {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5};
        try {
            // Attempt to access an element out of bounds
            int value = array[10];
            System.out.println("Value: " + value);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Don’t try buffer overflow
attacks in Java!");
        }
    }
}
```

Q12: 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.

```
package com.mycompany.q12;


public class CreditCard {
    private String customer;
    private String bank;
    private String account;
```

```
    private double limit;
    private double balance;

    public CreditCard(String cust, String bk, String acnt,
double lim, double initialBal) {
        customer = cust;
        bank = bk;
        account = acnt;
        limit = lim;
        balance = initialBal;
    }

    // Accessor methods
    public String getCustomer() { return customer; }
    public String getBank() { return bank; }
    public String getAccount() { return account; }
    public double getLimit() { return limit; }
    public double getBalance() { return balance; }

    // Method to charge a given price to the card, assuming
sufficient credit limit
    public boolean charge(double price) {
        if (price + balance > limit) {
            return false;
        }
```

```
        balance += price;
        return true;
    }

    // Method to make a payment to the card
    public void makePayment(double amount) {
        if (amount < 0) {
            throw new IllegalArgumentException("Amount
cannot be negative.");
        }
        balance -= amount;
    }
    public static void main(String[] args) {
        // Example usage
        CreditCard card = new CreditCard("John Doe",
"Bank of Java", "1234 5678 9012 3456", 5000, 1000);
        card.charge(200);
        System.out.println("Balance after charge: " +
card.getBalance());
        card.makePayment(150);
        System.out.println("Balance after payment: " +
card.getBalance());

        // This will throw an IllegalArgumentException
        card.makePayment(-50);
    }
```

}

الاسم:امةالملك محمد مظفر