1-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;
 }
```

Because of Recursive Call and Infinite Recursion

The sulotion :

```
public boolean charge(double price) {
    boolean isSuccess = super.charge(price);
    if (!isSuccess) {
        // Apply a one-time penalty of $5 if the initial charge fails
        isSuccess = super.charge(5); // Charge a $5 penalty
    }
    return isSuccess;
}
```

2- 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;
 }
```

subclasses like `PredatoryCreditCard` cannot directly access the `balance` field. Instead, they have to use the `charge` method (or other public methods) to interact with the `balance`.

Limited Visibility of `balance` (Private Visibility)

- If `super.charge(price)` fails (because the charge exceeds the credit limit), `isSuccess` will be `false`.

the penalty charge might also fail.

Recursive Call Issue

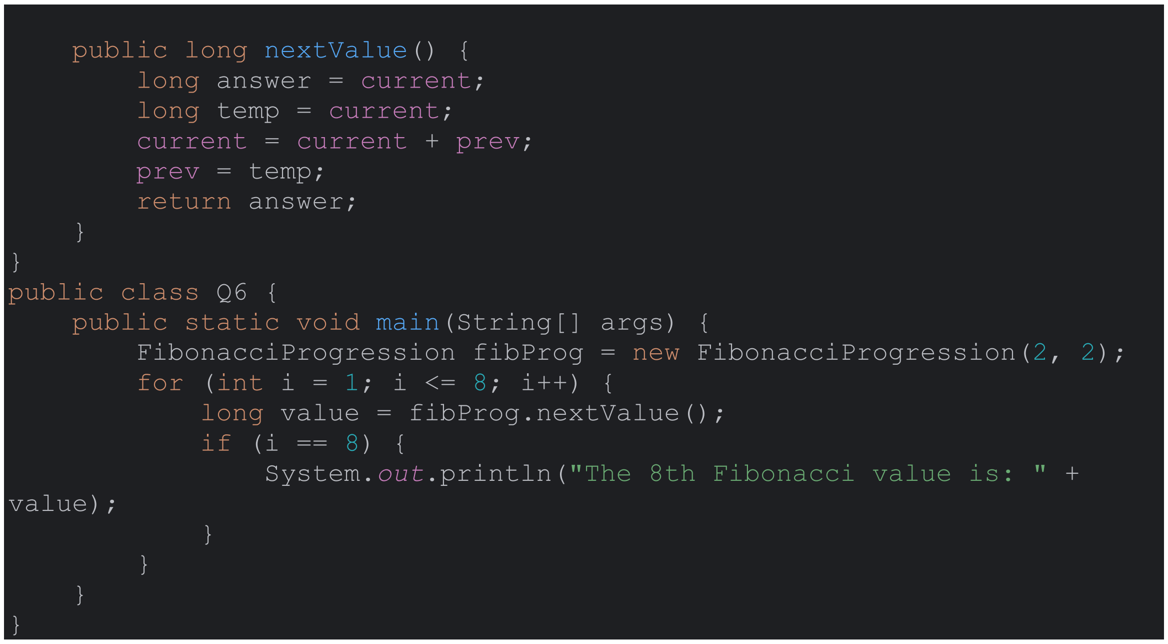
The Penalty Logic: While the intent of the `charge` method in `PredatoryCreditCard` is to apply a $5 penalty when a charge is denied, **this logic is flawed in two ways**

The sulotion :

```
public boolean charge(double price) {

    boolean isSuccess = super.charge(price);  // Try charging the
initial amount

    if (!isSuccess) {

        // Apply a one-time penalty of $5 if the initial charge fails

        isSuccess = super.charge(5);  // Charge a $5 penalty if the
initial charge fails

        if (!isSuccess) {

            System.out.println("Charge denied even with penalty.");

        }

    }

    return isSuccess;

}
```

3- 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.

```
class FibonacciProgression {
    private long current;
    private long prev;
    public FibonacciProgression(long first, long second) {
        current = first;
        prev = second;
    }
```



```
    public long nextValue() {
        long answer = current;
        long temp = current;
        current = current + prev;
        prev = temp;
        return answer;
    }
}
public class Q6 {
    public static void main(String[] args) {
        FibonacciProgression fibProg = new FibonacciProgression(2, 2);
        for (int i = 1; i <= 8; i++) {
            long value = fibProg.nextValue();
            if (i == 8) {
                System.out.println("The 8th Fibonacci value is: " +
value);
            }
        }
    }
}
```

4- 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?

- the `current` value will be: current=128×k
- We need to solve the inequality: 128×k≤9,223,372,036,854,775,807
    - Divide both sides by 128: k≤9,223,372,036,854,775,807/128
    - k≤72,057,594,037,927,297

Thus, the maximum number of calls to `nextValue()` before causing a `long` overflow is **72,057,594,037,927,297**.

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

لا لا يمكن لواجهتين ان تمتد كل منهما للاخرئ في لغة جافا وذلك لان مثل هذا السيناريو سيؤدي إلى حلقة لا نهائية أو اعتماد دائري في تسلسل الوراثة، وهو ما لا تسمح به جافا

6- 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 التعقيد في البحث عن الأساليب (التوزيع الديناميكي)

زيادة الحمل الناتج عن الوراثة من العديد من الفئات الأساسية

الارتباط الوثيق بين الفئات

زيادة الارتباط بين الفئات

الإفراط في استخدام الوراثة

صعوبة في التصحيح والتحليل

تأثير على قابلية القراءة والصيانة

احتمالية تكرار الكود

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?

زيادة التعقيد في الفئات

تكدس الأساليب في الفئة الأساسية

إفراط في استخدام الوراثة

الكفاءة في أداء البرنامج

تحديات في الصيانة

مشاكل في توسيع النظام

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[ ] 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.

Read 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( ).

```
                 Object
                 /    \
           Goat         Pig
             |             |
          milk(), jump()  eat(food), wallow()

               Horse
              /    \
        Racer        Equestrian
            |               |
          race()       weight, isTrained, trot(),    isTrained()
              height, color, run(), jump()
```

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?

No, the object referenced by `d` cannot be cast to the class `Racer` if `d` refers to an actual object of type `Equestrian`. Here's why:

Why Casting is Not Possible:

In Java, casting between classes in an inheritance hierarchy is allowed only when there is a direct parent-child relationship or a common ancestor. Since Equestrian and Racer are siblings (both extend Horse but are not related to each other directly), you **cannot cast an object of type Equestrian to Racer.**

11- 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 ArrayExample {
    public static void main(String[] args) {
        Scanner obj1 = new Scanner(System.in);
        Int [] myarr = {2,6,9};
        Int x = obj1.nextInt();
        Sout(“Enter an index “);

        try {
            sout(myarr[x]);}

        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Don’t try buffer overflow attacks in 
Java!");
        }
```

12- 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;
    }
```