Amal Al-Dubai / Software Engineering

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;  }
```

1. Recursive Call Without Stopping Condition:

The line charge(5); makes a recursive call to the charge method itself.

If super.charge(price) returns false, the method will attempt to impose a penalty charge of $5 by calling charge(5).

However, if the balance is already insufficient to process the penalty charge, super.charge(5) will also return false, and the recursive call will happen again. This creates an infinite recursion, leading to a StackOverflowError.

Correct Implementation

```
public boolean charge(double price) {
   boolean isSuccess = super.charge(price);
if (!isSuccess) {
      boolean penaltySuccess = super.charge(5);
 if (!penaltySuccess) {
      }
   }
   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;

1. Encapsulation Violation Risk

In the CreditCard class, balance is now private, meaning it cannot be directly accessed or modified outside the class.

2. Potential Logical Inconsistency

If the super.charge(5) call fails because the account balance is insufficient to cover the penalty, the penalty will not be applied. However, the method doesn't check whether the penalty was successfully imposed.

3. Implicit Dependency on super.charge

The implementation assumes that calling super.charge(5) is sufficient to handle the penalty logic. However, since balance is private, the PredatoryCreditCard class cannot directly verify or manipulate the balance to enforce specific rules related to penalties.

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.

```
FibonacciProgression fibonacci= new
FibonacciProgression(2,2);
fibonacci.printProgression(8);
```

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?

A long in Java is a 64-bit signed integer.

It ranges from -2^63 to 2^63 - 1, which is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

Formula:

value_n = start + (n-1)*increment

Final Answer:

You can make 72,057,594,037,927 calls to the nextValue method before causing a long-integer overflow, assuming the starting value is 0. If the starting value is non-zero, this number will be slightly lower.

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

No, two interfaces cannot mutually extend each other in Java.

Why?

Mutual extension would create a cyclical inheritance relationship, which is not allowed in Java. The compiler needs to fully resolve the structure of each interface during compilation, but a cyclic dependency makes it impossible to determine the members of the interfaces.

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.?

1. Runtime Performance Overhead
2. Increased Complexity
3. Reduced Flexibility
4. Code Maintenance Issues
5. Violation of Composition Over Inheritance Principle

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?

1. Reduced Code Reusability

2. Loss of Specificity

3. Maintenance Challenges

4. Performance Overhead

5. Difficulty in Extending the System

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?

The output

```
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    Horse
 ------------------------
                   |
            |       |
      Racer  Equestrian
```

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, d cannot be cast to the class Racer if it refers to an actual object of type Equestrian.

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 ArrayBoundsExample {
   public static void main(String[] args) {
   int[]x={1,2,3,4,5};
System.out.println("inpt index to print
(negative number to exit):");
Scanner input =new Scanner(System.in);
int a =input.nextint();
while(a>=0){
 try {
    System.out.println(x[a]);
       } 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;
      }
```