Exercises and Homework

1	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;
		}
		The method is flawed because it could result in an infinite loop. If the initial call to super.charge(price) fails, the method recursively calls itself with a penalty of 5. If the charge continues to fail, the recursion never stops, leading to an infinite loop .
		public boolean charge(double price) {
		boolean isSuccess = super.charge(price);
		if (!isSuccess) {
		super.charge(5);
		}
		return isSuccess;
		}

2	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

<table>
<tr><td></td><td></td><td>return isSuccess;
}

The issue arises if the penalty (5) would push the balance beyond the credit limit. In such cases, the penalty cannot be applied. The method should first check whether the penalty can be charged before calling super.charge(5).
public boolean charge(double price) {
boolean isSuccess = super.charge(price);

if (!isSuccess) {
// تحقق
if (canChargePenalty(5)) {
super.charge(5); // فرض العقوبة
}
}

return isSuccess;
}

// دالة مساعدة للتحقق
private boolean canChargePenalty(double penalty) {
return (balance + penalty) <= creditLimit;
}</td></tr>
<tr><td>3</td><td>R-2.6</td><td>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.
In R26</td></tr>
<tr><td>4</td><td>R-2.7</td><td>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?
In R27</td></tr>
</table>

5	R-2.8	Can two interfaces mutually extend each other? Why or why not? **No, two interfaces cannot mutually extend each other because this creates cyclic inheritance, which leads to ambiguity and conflicts in the implementation.**
6	R-2.9	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- Maintenance difficulty: Deep inheritance makes it harder to understand and manage the relationships between classes.** **2- Performance issues: Multiple layers of inheritance increase overhead due to method lookups.** **3- Code repetition: Deep hierarchies may lead to redundant or duplicate code**.
7	R-2.10	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- Lack of specialization: All classes extend the same base class, which limits flexibility.** **2- Cluttered base class: The base class may become overloaded with methods and attributes, making it harder to manage.**
8	R-2.11	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?

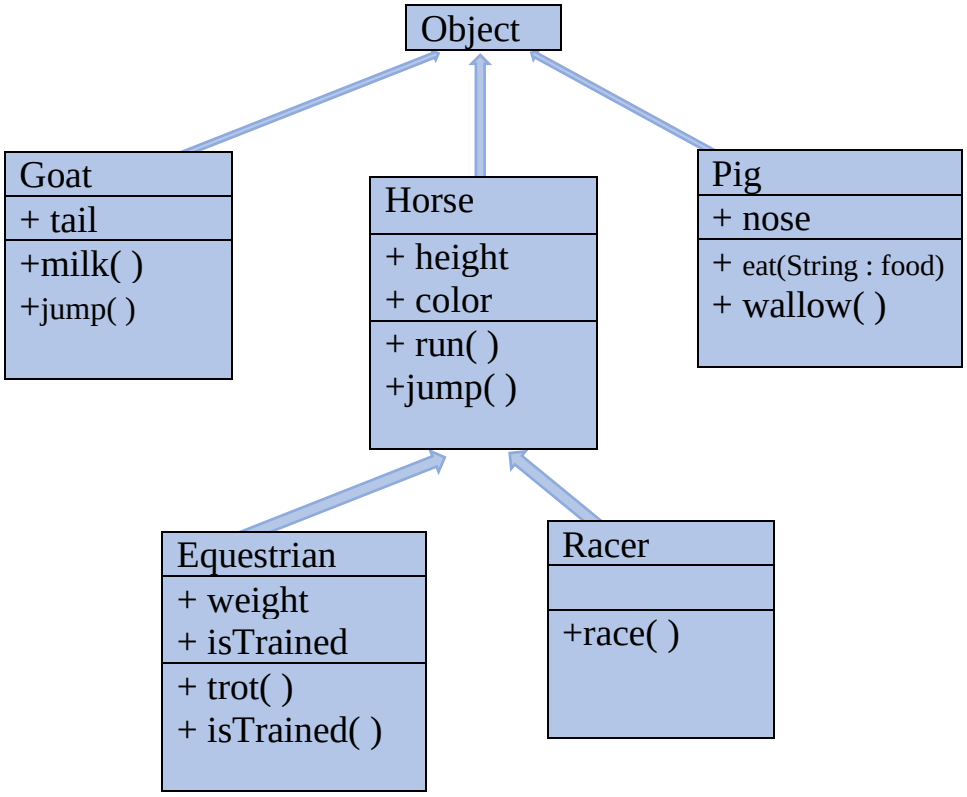
Chapter 2. Object-Oriented Design

R-2.11 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	R-2.12	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( ). 
10	R-2.13	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 , because Racer is not sub or supper for Equestrian . Equestrian can not be cast to class R-2.12 Racer .**

11	R-2.14	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!” **In R214**

12	R-2.15	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;`
		`}`