Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 9

Parametric Polymorphism and Information Hiding

November 23, 2018

Task 1

Consider the following Java method:

```
String concatenate(List<?> list) {
   String  result="";
   String separator="";
   if(list instanceof List<String>) {
      result="String:";
      separator=" ";
   }
   else if(list instanceof List<Integer>) {
      result="Integers:";
      separator="+";
   }
   for(Object el : list)
      result=result+separator+el.toString();
   return result;
}
```

A) This program is rejected by the Java compiler. Why?

B) Using the advice given by the Eclipse Java compiler (replace `List<...>` with `List<?>`), rewrite and compile the program. What are the results of executing the method passing each of the following:

- A list of strings containing only one element "word"?
- A list of Integers containing only one element Integer(1)?
- A list of Objects containing only one element (initialized by new Object())?

C) Is this behaviour consistent with what you would expect from the initial program? If not, how can you fix it?

D) What would happen if you tried to implement the different cases using method overloading instead of just one method. Why is this the case?

E) What happens if you compile and execute the initial program in C# ? Why?

Assume that we replace the wildcard by a method type parameter `T` to make it work in C#.

Task 2

A C++ template class can inherit from its template argument:

```
template <typename T>
class SomeClass : public T { ... }
```

A) Using this technique and given the following class definition

```
class Cell {
public:
   virtual void setVal(int x) { x_ = x; }
   virtual int value() { return x_; }
private:
   int x_;
}
```

write two template classes that can be used as "mixins" for class `Cell`

- `Doubling` - doubles the value stored in the cell.
- `Counting` - counts the number of times the value of the cell was read.

Do not use multiple inheritance. It should be possible to use the classes like this:

```
auto c = new Doubling<Counting<Cell>>(); // instantiation
c->setVal(5);
c->value(); // returns 10
c->numRead(); // returns 1
```

B) Describe how the instantiation above will look like.

C) How does this concept of mixins in C++ differ from Scala traits?

Task 3

Suppose that the following Java classes are part of a package, to which an external user cannot add classes.

```
public abstract class BankAccount {
   ... boolean importantCustomer=false;
   ... int amount=0;
   ... final int maxDebit=1000;

   /// invariant amount >= -maxDebit &&
   ///    !importantCustomer => amount>=0 &&
   ///    importantCustomer <=> this instanceof RichCustomer

   ... void deposit(int amount);
   ... void withdraw(int amount);
}

public final class PoorCustomer extends BankAccount {
   ... void  deposit(int amount) {
      if(amount>=0)
         this.amount+=amount;
   }
   ... void  withdraw(int amount) {
      if(amount<=this.amount)
         this.amount-=amount;
   }
```

```
}

public final class RichCustomer extends BankAccount {
   public RichCustomer() {importantCustomer=true;}
   ... void  deposit(int amount) {
      if(this.amount+amount >= -maxDebit)
         this.amount+=amount;
   }
   ... void  withdraw(int amount) {
         if(-maxDebit<=this.amount-amount)
            this.amount-=amount;
   }
}
```

Provide the most permissive access modifiers for each field and method, such that the class invariant cannot be broken from outside the package. Assume that no integer over/underflow occurs.

In Scala, a class can be declared as sealed. That means that the class can be extended only by classes written in the same `.scala` file. Suppose that the class `BankAccount` is declared as sealed, and `PoorCustomer` and `RichCustomer` are part of the same `.scala` file. Does this allow you to choose more permissive access modifiers?

Task 4

From a previous exam

Consider the following Java program consisting of two packages:

```
1  package A;
2
3  public abstract class Person {
4      _______ int tickets = 0;
5      _______ final int maxTickets = 3;
6
7          /// invariant 0 <= tickets <= maxTickets
8
9      _______ abstract void buy(int t);
10 }
11
12 public class Buyer extends Person {
13     _______ void inc(int t) {
14         if (this.tickets+t <= this.maxTickets) this.tickets += t;
15     }
16     _______ void buy(int t) { if (t >= 0) inc(t); }
17 }
18
19
20
21 package B;
22 import A.*;
23
24 public class SmartBuyer extends Buyer {
25     _______ void inc(int t) { this.tickets += t; }
26 }
27
28 public class Main {
29     public static void main(String args[]) {
30         Buyer b = new SmartBuyer();
31         b.buy(9);
32     }
33 }
```

A) Provide the *most restrictive* access modifiers for the fields `tickets` and `maxTickets` and the methods `inc()` and `buy()` such that the program is still accepted by the compiler.

B) With respect to the access modifiers that you provided in part **A**, add code to the class `SmartBuyer` such that the execution of the `main()` method of the class `Main` breaks the invariant of the class `Person`.

Task 5

Consider the following Java code:

```
package p;

public final class List {
   ///invariant 1: The list starting at head is acyclic
   ///invariant 2: The list starting at head is non-decreasing

   public void prepend(int x){
      if (head==null || x <= head.getValue())
         head = new Node(x,head);
   }

   public Node getHead(){ return head; }
   public Node head = null;
}

public final class Node {
   Node(int x, Node n) {
      value = x;
      next = n;
   }

   public Node getNext(){ return next; }
   public int getValue(){ return value; }
   private Node next;
   private int value;
}
```

Assuming that we cannot modify the classes `List` and `Node`, we would like to see whether or not the invariants can be broken, either by adding classes to package `p`, or by clients outside of package `p`. Assume reflection is not used at all.

A) Can invariant 1 be broken by adding clients outside of package `p`? If yes, show code, that when run ends in a state in which the invariant is broken; if not explain why.

B) Can invariant 1 be broken by adding classes to package `p`? If yes, show code, that when run ends in a state in which the invariant is broken; if not explain why.

C) Can invariant 2 be broken by adding clients outside of package `p`? If yes, show code, that when run ends in a state in which the invariant is broken; if not explain why.

D) Can invariant 2 be broken by adding classes to package `p`? If yes, show code, that when run ends in a state in which the invariant is broken; if not explain why.

Task 6

Consider the following Java code:

```
public class Hour {
   public int h=0;
}

public class Time {
   private Hour hour=new Hour();
   private int m=0;
   /// invariant hour.h>=0 && hour.h<24

   public void setHour(int h) {
      if(h>=0 && h<24) this.hour.h=h;
   }

   public Hour getHour() {return hour;}
}
```

A) Provide an example that breaks the invariant of `Time` without changing the code above and without using reflection.

B) There are two immediate ways to fix the problem. In one of them, signatures of methods are modified, while in the other they are not. What are these ways of fixing the problem?

C) Clearly, we would prefer to keep the signatures the same as before. Are there any drawbacks to this approach?

D) Would it be possible to introduce an interface with no mutator methods and use it to solve the problem? Explain how this approach would look and whether there would still be a way to break the invariant.

Task 7

Consider the following Java programs:

Program 1	Program 2	Program 3	Program 4

`package A1;`
`public class X {`
`  int x;`
`}`
	`package A1;`
	`public class X {`
	`  protected int x;`
	`}`
		`package A1;`
		`public class X {`
		`  private int x;`
		`}`
			`package A1;`
			`public class X {`
			`  protected int x;`
			`}`

`package A2;`
`import A1.X;`
`class Y extends X`
`{`
`  int f(X v) {`
`  return v.x;`
`  }`
`}`
	`package A2;`
	`import A1.X;`
	`class Y extends X`
	`{`
	`  int f(X v) {`
	`  return v.x;`
	`  }`
	`}`
		`package A2;`
		`import A1.X;`
		`class Y extends X`
		`{`
		`  int f(X v) {`
		`  return v.x;`
		`  }`
		`}`
			`package A2;`
			`import A1.X;`
			`class Y extends X`
			`{`
			`  int f() {`
			`  return this.x;`
			`  }`
			`}`

Only one of these programs compiles. Which one? Why are the other programs rejected?

You can refer to the Java Language Specification rule 6.6.2.1 for more detailed information about the `protected` access modifier.