Exercise 8

Information hiding, encapsulation, aliasing

1) Suppose that we have a language with the information hiding rules of Java, but with structural subtyping. What should be the subtyping relations between the following three classes?

```
class A {int foo();}
class B {protected int foo();}
class C {public int foo();}
```

2) 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?

3) Consider the following class definitions (in the same package),

```
class A {
     int x=0;
     void print() {System.out.println("Class A:"+x);}
     void setX(int v) {x=v;}
}

public class B extends A {
     int x=0;
     void print() {System.out.println("Class B:"+x);}
     protected void setX(int v) {x=v;}

}
```

as well as the following two clients:

```
void foo(A obj) {
      obj.x=10;
      obj.print();
}
```

```
void bar(A obj) {
    obj.setX(10);
    obj.print();
}
```

What happens if we execute `foo`? What about `bar`? Explain in detail. In what sense is `bar` preferable to `foo`?

4) Consider the class `Hour`, defined as follows:

```
public class Hour {
    protected int h=0;
    //invariant h>=0 && h<24

    public set(int h) {
        if(h>=0 && h<24) this.h=h;
    }
}
```

What is the external interface of `Hour`?

Can we extend the code, without changing the class, in a way that the invariant is broken? If yes, provide an example, and propose how to fix the class.

5) Suppose that the following 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(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 invariant cannot be broken from outside the package.

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?

6) Java allows an object of a class `C` to access the *private* fields of other objects declared in `C`. Discuss the resulting level of information hiding, its advantages and limits, and provide some examples.

What is the policy concerning the visibility of *protected* fields of other objects?

7) 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;}
}
```

Provide an example that breaks the invariant of `Time`.

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?

8) The Java API 1.6 of class `String` states:

"Strings are constant; their values cannot be changed after they are created. (…) Because String objects are immutable they can be shared."

Consider the following example:

```
String prettyprint(List<T> list) {
  String result = "";
  for(int i=0; i<list.size(); i++)
    result = result+list.get(i).toString()+",";
  return result;
}
```

Discuss the advantages and the drawbacks of immutable strings.