Exercise Sheet 6

1. Look at the following program:

```
class Motor {
   boolean isOK() { return true; }
   void start() { /*...*/ }
   ...
}
class Wheel {
   void deflate() { /*...*/ }
   ...
}
class MotorTrouble extends Exception {
   public Motor motor;
   public MotorTrouble( Motor m ) {
        motor = m;
   }
}
public class Car {
   public Motor engine;
   public Wheel[] wheels;

   /*@ invariant engine.isRunning() ==>
     @ (\forall int i; i >= 0 && i < wheels.length;
     @          wheels[i] != null && wheels[i].isOK())
     @*/

   public Car( Motor m, Wheel[] w ) {
        engine = m;
        wheels = w;
   }
   public Motor getMotor() {
        return engine;
   }
   public Wheel[] getWheels() {
        return wheels;
   }
   public void start() throws MotorTrouble {
      if( engine == null || !engine.isOK() ) {
         throw new MotorTrouble( engine );
      } else {
         engine.start();
      }
   }
}
```

   a. What aliasing problems can arise in the example program?
   b. Write example code for every problem where the invariant can be destroyed.
   c. Change the code of `Car` in a way so that there are no more aliasing problems.

2. In lecture 5 we have seen what is needed to preserve the consistency of invariants in Java. Now we want to allow invariants that can refer to private and default access fields.

   a. Give an example program showing that the existing obligations are not sufficient.

   b. Strengthen the obligations in a way so that consistency is preserved again.

3. **Encapsulation question from previous exam!**
   This example addresses the relation between encapsulation techniques and security aspects. Given the following scenario: A system environment, represented by the object of type `Environment`, manages what people have access to secure parts of the system. The ID of the persons are stored as an `int` in the class `Authorization` . `Environment` and `Authorization` are implemented in the following way:

```
package System;
public interface Environment {
   public void insertAuthorization ( Authorization b);
   public Authorization getAuthorization();
}
package System;
public class Authorization {
   private int[] ids;
   public Authorization() { ids = new int[5]; }
   protected void setIDs( int[] p ) {
      ids = p; }
   public int[] getIDs() { return ids; }
}
```

The interaction between `Environment` and `Authorization` looks like the following:

- Objects of type `Authorization` can be created by an arbitrary class and can be transfered to the system environment with the method `insertAuthorization`.
- `insertAuthorization` saves the transfered reference and stores the ID of the registered person into the field IDs of the `Authorization` object using the method `setIDs`. (Keep in mind, that `Environment` and `Authorization` are declared in the same package!)
- An arbitrary user of the system can fetch the IDs of the registered people with the methods `getAuthorization` and `getIDs` accessing them read only. For example, to make comparisons between ids.

The interaction between `Environment` and `Authorization` is called a secure system, if no class outside of the package `System` is allowed to modify the IDs stored into the `Authorization` object.

Exercise:

a. The above implementation is not secure. Describe how an attacker can manipulate the list of `ids` using the method `getIDs.` In this case an attacker is a class declared outside of the package system.

b. Implement your solution for question **a** as method
`public static void attack(Environment u) {...}`
in class `Attack` of package `Attacker.`

c. Explain how to modify the implementation of class `Authorization`, to prohibit the attack. The modified `Authorization` class still has to allow the read only interaction described above.
The interface `Environment` as well as the implementation must not be modified.

d. Describe how an attacker could manipulate the list of IDs **without using** the method `getIDs.`

e. Implement your solution for question **d** as method
`public static void attack( Environment  u) {...}`
in a class `Attack` in the package `Attacker.`

f. Explain how to modify the implementation of class `Authorization` to prohibit the attack from question **d**. There are the same requirements as in question **c**.

g. **[Homework 7]**
How can the Universe Type system be used to prevent such problems?
Give an annotated source code of the classes `Environment` and `Authorization`. Assume that the objects `Environment` and `Authorization` are stored in the same universe. Change the implementation of the methods of both classes as needed.