Übungsblatt 6

1. Betrachte folgendes Programm:

```
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. Welche Aliasing-Probleme können in dem Beispielprogramm auftreten?
   b. Schreibe für jedes Problem Code, durch den die Invariante verletzt wird.
   c. Ändere den Code von Car so, dass keine Aliasing-Probleme mehr auftreten.

2. In der Vorlesung haben wir gesehen, was zur Sicherstellung der Konsistenz von Invarianten in Java geprüft werden muss.
   Siehe Vorlesung 5, Folie 41, "Invariants for Java".
   Wir wollen nun erlauben, dass sich Invarianten auf private und default access Felder beziehen dürfen.

   a. Gib ein Beispiel-Programm an, welches zeigt, dass die bestehenden Prüfungen nicht ausreichen.

   b. Verstärke die Prüfungen, damit Konsistenz wieder sichergestellt wird.

3. **Klausurbeispiel vom letzten Jahr!** Kapselung
   Diese Aufgabe behandelt den Zusammenhang zwischen Kapselungstechniken und Sicherheitsfragen.
   Gegeben ist folgendes Szenario: Eine Systemumgebung, repräsentiert durch ein Objekt vom Typ `Umgebung`, verwaltet, welche Personen Zugriff auf geschützte Systemteile haben. Deren Kennungen (ID der Personen) werden als `int`s in einem Objekt der Klasse `Befugnis` gespeichert. `Umgebung` und `Befugnis` sind wie folgt implementiert:

```
package System;
public interface Umgebung {
   public void einfuegenBefugnis(Befugnis b);
   public Befugnis holeBefugnis();
}
package System;
public class Befugnis {
   private int[] kennungen;
   public Befugnis() { kennungen = new int[5]; }
   protected void setzeKennungen( int[] p ) {
      kennungen = p; }
   public int[] holeKennungen() { return kennungen; }
}
```

   Das Zusammenspiel zwischen `Umgebung` und `Befugnis` ist wie folgt:
   - Objekte vom Typ `Befugnis` können von beliebigen Klassen erzeugt und mit der Methode `einfuegenBefugnis` an die Systemumgebung übergeben werden.
   - `einfuegenBefugnis` speichert die übergebene Referenz und trägt die Kennungen der zugelassenen Personen in das `kennungen`-Feld des `Befugnis`-Objekts ein, indem es die Methode `setzeKennungen` aufruft. (Beachten Sie, dass `Umgebung` und `Befugnis` im gleichen Paket deklariert sind!)
   - Beliebige Nutzer des Systems können sich mittels der Methoden `holeBefugnis` und `holeKennungen` die Kennungen der zugelassenen Personen holen, um **lesend** auf diese zuzugreifen, also um diese z.B. mit anderen Kennungen zu vergleichen.

   Wir nennen das Zusammenspiel zwischen `Umgebung` und `Befugnis` ein *sicheres System*, wenn keine Klasse ausserhalb des Pakets `System` die in einem `Befugnis`-Objekt gespeicherten Kennungen verändern kann.

Aufgaben:

a. Mit der obigen Implementierung ist das System nicht sicher. Beschreiben Sie, wie ein Angreifer **unter Verwendung** der Methode `holeKennungen` die Liste der Kennungen manipulieren könnte! Unter einem Angreifer verstehen wir eine Klasse, die ausserhalb des Pakets System deklariert ist.

b. Implementieren Sie Ihre Lösung zu Teilaufgabe a) als Methode
```
public static void angreifen(Umgebung u) {...}
```
einer Klasse `Attacke` im Paket `Angreifer`!

c. Geben Sie an, wie man die Implementierung der Klasse `Befugnis` ändern müsste, um den Angriff aus Teilaufgabe a) zu verhindern! Die modifizierte `Befugnis`-Klasse muss jedoch die Punkte 1–3 des oben beschriebenen Zusammenspiels weiterhin zulassen!
Auch die Schnittstelle `Umgebung` sowie Ihre Implementierung müssen von der Änderung unberührt bleiben.

d. Beschreiben Sie, wie ein Angreifer **ohne Verwendung** der Methode `holeKennungen` die Liste der Kennungen manipulieren könnte!

e. Implementieren Sie Ihre Lösung zu Teilaufgabe d) als Methode
```
  public static void angreifen(Umgebung u) {...}
```
einer Klasse `Attacke` im Paket `Angreifer`!

f. Geben Sie an, wie man die Implementierung der Klasse `Befugnis` ändern müsste, um den Angriff aus Teilaufgabe d) zu verhindern! Es gelten die gleichen Rahmenbedingungen wie bei Teilaufgabe c).

g. Wie kann das Universe Typsystem verwendet werden, um diese Probleme zu umgehen?
Geben Sie den annotierten Quellcode für die Klassen `Umgebung` und `Befugnis` an. Gehen Sie davon aus, dass `Umgebung`- und `Befugnis`-Objekte sich im gleichen Universe befinden. Nehmen Sie notwendige Änderungen an den Implementierungen der Methoden der beiden Klassen vor.