Exercise 9

Aliasing, encapsulation of object structures, read only types
25th November

In-class Assessment: One or more questions from this sheet will be used for the in-class assessment. No notes are allowed during the assessment.

1) Data structures often intentionally share aliases. For instance, consider the following `ArrayList` class:

```
class ArrayList<T> {
  private T[] elements=…;
  private int LastEl=0;
  public T get(int i) {return elements[i];}
  public void add(T el) {elements[intLastEl++]=el;}
}
```

Imagine that this class is extended as follows

```
class Coordinates {
  int x, y;
  public Coordinates(int xx, int yy) {x=xx; y=yy;}
}

class CList extends ArrayList<Coordinates> {
  // invariant \forall el \in elements : el.x>el.y
  public void add(Coordinates el) {
    if(el.x>el.y) super.add(el);
  }
}
```

Write a program that breaks the invariant of `CList`. How can we fix this problem? Is it possible to fix it without allocating new objects (either directly or indirectly), that is, without consuming additional memory? What new problems might arise from your changes? Discuss the benefits and the drawbacks of using alias sharing in data structures.

2) In Question 7 of last week's sheet we were able to break the invariant because of alias leaking.

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

public class Time {
    private Hour hour;
    //invariant hour.h>=0 && hour.h<24

    public Time(Hour hour) {this.hour=hour;}

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

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

Modify class `Hour` by using read-only interfaces. Find an example that still breaks the invariant, and explain why such a solution is not satisfactory in terms of safety.

3) Consider the following C++ class:

```
class Person
{
      int money;
      Person *spouse;

public:
      void f () const;
      Person (int m, Person *s)
      { if (!s) spouse = 0;
        else { spouse = s; s->spouse = this; }
        money = m;
      }
};
```

Method `f` promises not to make any changes to its target object. Show that this claim can still be violated by a definition of `f`, without using casting and without introducing any local variables.

4) The intuition behind a `pure` method is that its execution effects are not observable by the client. This essentially means that the result of any other method call or field read inside client code would not be affected by a pure method execution. One way to formalize this property is to require that the execution of a pure method does not change the program heap.

- Provide proof obligations that guarantee the purity of a method, according to this requirement. Can you define an analogous notion for constructors?
- Class `Set` represents a set of integers. Method `Set allLessThan(int bound)` (in class `Set`) returns a freshly-allocated instance of class `Set` that contains all elements of the original set that are smaller than `bound`..
    - Even though the method `allLessThan` does not change the behavior of other methods, it is not pure, according to our definition. Why?
    - How can the provided definition of purity be relaxed to allow declaration of the method `allLessThan` as pure, without violating the intuition above?
    - Provide proof obligations that guarantee purity of a method according to your relaxed definition.
    - Can you define an analogous notion for constructors?

5) Consider the following classes:

```
class A {
  readwrite StringBuffer n1=…;
  readonly StringBuffer n2=…;
}

class B {
  readwrite A x;
  readonly A y;
  public B(readwrite A x, readonly A y) {
    this.x=x;
    this.y=y;
  }
}
```

Check which programs typecheck and explain why they do or do not typecheck.

Program 1	**Program 2**	**Program 3**

`readwrite A obj=new A();`
`readonly B obj2=`
`  new B(obj, obj);`
`readwrite StringBuffer v=`
`  obj2.y.n1;`
	`readwrite A obj=new A();`
	`readwrite B obj2=`
	`  new B(obj, obj);`
	`readwrite StringBuffer v=`
	`  obj2.y.n1;`
		`readwrite A obj=new A();`
		`readwrite B obj2=`
		`  new B(obj, obj);`
		`readwrite StringBuffer v=`
		`  obj2.x.n1;`

Program 4	**Program 5**	**Program 6**

`readonly A obj=new A();`
`readonly A obj2=new A();`
`readwrite B obj3=`
`  new B(obj, obj2);`
`readwrite StringBuffer v=`
`  obj3.y.n1;`
	`readwrite A obj=new A();`
	`readonly A obj2=new A();`
	`readwrite B obj3=`
	`  new B(obj, obj2);`
	`readonly StringBuffer v=`
	`  obj3.y.n1;`
		`readwrite A obj=new A();`
		`readonly A obj2=new A();`
		`readwrite B obj3=`
		`  new B(obj, obj2);`
		`readonly StringBuffer v=`
		`  obj3.y.n2;`