Exercise 10

Ownership type system

3rd December

1. Consider the following method signatures:

```
peer Object foo(any String el);
peer Object foo(rep String el);
rep Object foo(any String el);
any Object foo(peer String el);
rep Object foo(peer String el);
```

   Find all the valid pairs of signatures such that one overrides the other.

2. 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 are correct and explain why.

Program 1
`readwrite A obj= new A();`
`readonly B obj2= new B(obj, obj);`
`readwrite StringBuffer v=obj2.y.n1;`
	Program 2
	`readwrite A obj=new A();`
	`readwrite B obj2=new B(obj, obj);`
	`readwrite StringBuffer v=obj2.y.n1;`

Program 3
`readwrite A obj=new A();`
`readwrite B obj2=new B(obj, obj);`
`readwrite StringBuffer v=obj2.x.n1;`
	Program 4
	`readonly A obj=new A();`
	`readonly A obj2=new A();`
	`readwrite B obj3=new B(obj,obj2);`
	`readwrite StringBuffer v=obj3.y.n1;`

Program 5
`readwrite A obj=new A();`
`readonly A obj2=new A();`
`readwrite B obj3=new B(obj, obj2);`
`readonly StringBuffer v=obj3.y.n1;`
	Program 6
	`readwrite A obj=new A();`
	`readonly A obj2=new A();`
	`readwrite B obj3=new B(obj,obj2);`
	`readonly StringBuffer v=obj3.y.n2;`

3. Look at the following program:

```
class ArrayList {
      protected int[] array;
      protected int next;

      public void add(int i) {
        if( next==array.length ){
          resize( );
        }
        array[ next ] = i;
        next++;
      }

      public int[] getElems() {
        return array;
      }

      public void setElems(int[] ia){
        array = ia;
        next = ia.length;
      }
      protected void resize() {
        if( next==array.length ) {
          int[] oa = array;
          array = new int[2*oa.length];
          System.arraycopy
           (oa, 0, array, 0, oa.length );
        }
      }

      public String toString() {
        if( array.length == 0 ) return "[]";
        StringBuffer buf =
          new StringBuffer("[ " + array[0]);

        for( int i=1; i < next; ++i ) {
          buf.append(", " + array[i]);
        }
        buf.append(" ]");
        return buf.toString();
      }
}
```

a. What aliasing problems can arise in the example program?
b. Write example code for every problem.
c. Change the code of ArrayList in a way that guarantees that there are no more aliasing problems.
d. Annotate ArrayList with appropriate ownership type modifiers

4. Annotate the following program with appropriate ownership type modifiers to maximize the buffer, the producer, and the consumer encapsulation:

```
class Producer {
  int[] buf;
  int n;
  Consumer con;
  Producer() {
    buf = new int[10];
  }
  void produce(int x) {
    buf[n] = x;
    n = (n+1) % buf.length;
  }
}
```

```
class Consumer {
  int[] buf;
  int n;
  Producer pro;
  Consumer(Producer p) {
    buf = p.buf;
    pro = p;
    p.con = this;
  }
  int consume() {
    n = (n+1) % buf.length;
    return buf[n];
  }
}
```

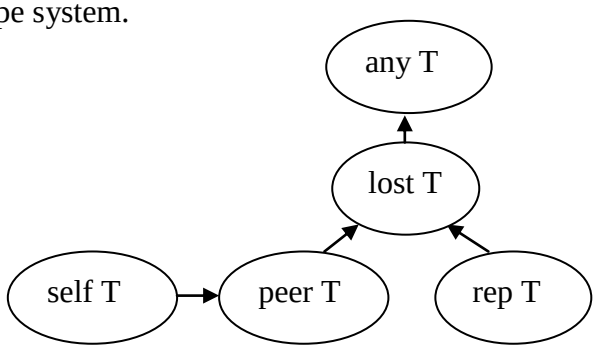
```
class Context {
  Producer p;
  Consumer c;

  Context(){
    p = new Producer();
    c = new Consumer(p);
  }

  public void run() {
    for(int i=-5; i <=5; ++i){
      p.produce(i);
      if(i%2 == 0)
        c.consume();
    }
  }
}
```

5. **Encapsulation question from a previous exam!**
   The *Universe type system* allows the following ownership modifiers **peer**, **rep**, **self**, **lost**, and **any** - to structure the object store and to restrict how references can be passed and used. We want to extend the *Universe type system* by adding one more modifier **down**. This modifier is introduced to denote references to objects in the same context as **this** or in the context (*transitively*) owned by an object in the same context as **this**.

   - Redraw the subtype relation diagram below to include the newly introduced type of the universe type system.



   - Define the most specific (in terms of the context information it conveys) type combinator function ►by filling the table below (first argument: left-most cell of the rows, second argument: top-most cell of the columns).

     Recall that the type combinator function ► is used, in particular, to determine the owner of an object referenced by a field access. More exactly, if the ownership

modifier of x is T_x and the ownership modifier of a field f is T_f, then the ownership modifier assigned to the field access $x.f$ is determined as $T_x \blacktriangleright T_f$.

▶	**peer**	**rep**	**lost**	**any**	**down**
self					
peer					
rep					
lost					
any					
down					

a. Define type checking rules for field update.

6. (harder) Consider the following code:

```
public class List{

   …

   public void addFirst(int x) {
      head = new Node(x,head);
   }

   public List clone(){
      return new List(this);
   }

   private List(List other){
      head = null;
      Node p = null;
      for (Node n=other.head;n!=null;n=n.next){
            Node h = new Node(n.val,null);
            if (p!=null)
                  p.next=h;
            else
                  head = h;
            p = h;
      }
   }

   rep Node head;
   private class Node{
      Node( int val, Node next){
            this.next = next;
            this.val = val;
      }
      Node next;
      int val;
   }
}
```

```
class Client{
   void f(any List list){
      this.list = list.clone();
      this.list.addFirst(42);
   }

   rep List list;
}
```

a. Can you annotate these classes with ownership?
b. What problems do you encounter – are these because of an aliasing issue with the implementation or the type system?
c. Can you think of a way to modify the ownership type system to allow for this example to be typed:
   i. What kind of topological property would you like to describe?
   ii. What rules do you need to preserve this property? Think about field reads and field writes
   iii. How does this property relate to the already defined ones (**rep**,**peer** etc)? can you suggest rules for casting?