Exercise 10

Ownership type system

1. 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();
}
```

```
}
```

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

2. 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();
    }
  }
}
```

3. Use the Universe type system to guarantee the encapsulation of *Entry*, *LinkedList*, *ReadIterator*, and *DeleteIterator*. Additionally, annotate the methods with appropriate pure modifiers. If needed, change the implementation in a way such that Universe typing is possible. Provide arguments to motivate necessity of the implementation changes.

For this task you are allowed to:

- Add new methods to the classes LinkedList and DeleteIterator.
- Assume that each object has a field "owner" that can be mentioned in pre/post-conditions and invariants.
- Add pre/post-conditions and invariants that may mention field owner, to the classes LinkedList and DeleteIterator.
- Use casting from **any** to **peer** and **rep** if it is possible to deduce from pre/post-conditions and invariants that it will not result in a runtime exception.

```
class Entry {
  Object element; Entry previous, next;
  Entry( Object o, Entry p, Entry n )
   {element=o; previous=p; next=n;}
}
```

```
class ReadIterator {
  protected Entry current, header;

  public ReadIterator( Entry h )
    {current = h; header = h;}

  public boolean hasNext()
    {return current.next != header;}

  public void moveNext()
    {current = current.next;}

  public Object element()
    {return current.element;}
}
```

```
class DeleteIterator extends ReadIterator {

public DeleteIterator(Entry h)
  {super( h);}

public void delete() {
  if( current.previous != null )
    current.previous.next = current.next;
  if( current.next != null )
    current.next.previous=current.previous;
  current = current.next;
  }
}
```

```
public class LinkedList {
  private Entry header;
  private int size;

  public LinkedList() {
    Init();
  }

  public Init() {
  // the header is a dummy that is never null
    header =
    new Entry(null, null, null);
    header.next = header;
    header.previous = header;
    size = 0;
  }

  public void add(Object o){
    Entry newE=
      new Entry(o,header,header.next );
    header.next.previous = newE;
    header.next = newE;
    ++size;
  }

  public Object get( int idx ) {
    if( idx > size ) return null;
    Entry e = header.next;
    for( int i=0; i<idx; ++i ) {
      e = e.next;
    }
   return e.element;
  }

  public ReadIterator getReadIterator() {
   return new ReadIterator( header );
  }

  public DeleteIterator getDeleteIterator() {
   return new DeleteIterator( header );
  }
}
```

4. The linked list from the previous example is extended with the method merge. The method moves all nodes from one list to the other.

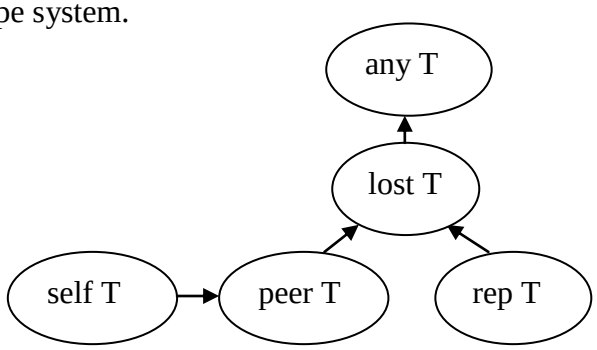
- Explain why the proposed version of merge can't be properly annotated with Universe type system.
- Change the implementation in a way such that Universe typing is possible and annotate the updated method.

```
public void merge(LinkedList other) {
  if (other.header.next == other.header)
    return;

  Entry first = other.header.next;
  Entry last = other.header.previous;

  header.previous.next = first;
  last.next = header;

  first.previous= header.previous;
  header.previous = last;

  other.Init();
}
```

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**.

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



b. 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►T_f.

►	**peer**	**rep**	**lost**	**any**	**down**
self					
peer					
rep					
lost					
any					
down					

c. Define type checking rules for field update.