Question 1 (24 points)

Our friend Willie Wazoo has created the code below to implement a finite string bag (FSB). A FiniteStringBag is a set that allows duplicate values, also called a multiset. The representation uses a fixed length array to hold the elements and has an associated integer variable to record the size of the bag (the number of elements being used in the array).

```
class FiniteStringBag {
       // Rep: items is a fixed length array
       // items[0..size-1] contains Strings
       // size <= items.length
       // Entries cannot be null. There may be multiple copies
       // of the same string in the FiniteStringBag (FSB).
       private String[] items;
       private int size;

       //Construct new FSB with given capacity.
       // Requires: capacity > 0
       public FiniteStringBag(int capacity) {
              this.items = new String[capacity];
              this.size = 0;
       }

       /** Return capacity of this FSB */
       public int capacity() { return items.length; }
       /** Return current size of this FSB*/
       public int size() { return size; }

       /** Return item at position i of this FSB */
       public String get(int i) {
              if(i >= size)
                     throw new IndexOutOfBoundsException();
              return items[i];
       }
       public void add(String s) {
              if(size == items.length)
                     throw new BufferOverflowException();
              items[size] = s;
              size = size + 1;
       }
       /** Return whether s is located in this FSB */
       public boolean contains(String s) {
              if (size == 0) return false;
              for(int i = 0; i < size; i++) {
                     if(items[i].equals(s))
                            return true;
              return false;
       }
       // delete strings with length > n
       public void deleteLongStrings(int n) {
              int k = 0;
              while (k < size) {
                     if (items[k].length() > n) {
                            items[k] = items[size-1];
                            size = size-1;
                     } else {
                            k = k + 1;
                     }
              }
       }
```

```
        // additional methods to be added later...…
```

a) (4 points)
Give a suitable abstraction function (AF) for this class relating the representation to the abstract value of a finite string bag.

b) (5 points)

`FiniteStringBag` lacks a `checkRep()` method. Write a `checkRep()` method for this class. The method should check the representation as described in the comments at the start of the class definition.

c) (9 points)
Describe three separate, distinct “black box” tests for the deleteLongStrings method. You don’t need to write Java or Junit code. Just give a brief clear description of the test.

d) (2 points)
Are there any potential problem from representation exposure with the FiniteStringBag as it written above? Why or why not? Be brief.

e) (4 points)

We would like to add an observer method to FiniteStringBag. Willie proposes the following method

```
// return the current strings in this FSB to the caller
public String[] getItems() {
      return items;
}
```

Willie’s colleague Ima Hacker points out that there are two problems with this method. What are they?

Question 2. (18 pts)

Consider the code.

```
class A {
   void m(A a) { System.out.println("AA"); }
   void m(B a) { System.out.println("AB"); }
   void m(C a) { System.out.println("AC"); }
}
class B extends A {
   void m(A a) { System.out.println("BA"); }
   void m(B a) { System.out.println("BB"); }
   void m(C a) { System.out.println("BC"); }
}
class C extends B {
   void m(A a) { System.out.println("CA"); }
   void m(B a) { System.out.println("CB"); }
   void m(C a) { System.out.println("CC"); }
}

A a1 = new A();
A a2 = new B();
A a3 = new C();
B b1 = new B();
B b2 = new C();
C c1 = new C();
```

Fill in each box in the above table with the output of the corresponding invocation. For example, fill in the cell for row headed by **`b1`** and column headed by **`a2`** with the output of **`b1.m(a2)`**.
Hint: There is a pattern to t answers in the table.

	a1	a2	a3	b1	b2	c1
a1						
a2						
a3						
b1						
b2						
c1						

Question 3 (6 points)

Please keep your answers short and clear.

a) (2 points)
In Java, unchecked exceptions are used to convey special results to the calling method (e.g., IllegalArgumentException). (True or False)

b) (2 points)
Suppose we have a class A with a method m:

```
    class A {
        public T m(S x) { ... }
    }
```

Now suppose we create a class B that is a subclass of A with its own method m that is supposed to override the one from class A:

```
    class B extends A {
        public T1 m(S1 x) { ... }
    }
```

If we want class B to be a true subtype of class A, what are the possible subtype/supertype relationships between m's types T and S in A and types T1 and S1 in B?

c) (2 points)
We want to refactor the following method so that it uses generic types.

```
// effects: dst is a copy of src without duplicates.
// modifies: dst. src is not modified.
static void removeDups(Collection src, Collection dst);
```

Fill in the blanks in the following signature with either `extends` or `super`.

```
static void removeDups(Collection<? ___________________________ T> src,

                          Collection<? __________________________  T> dst);
```

Question 4 (14 points)

Consider the following class hierarchies:

```
class Mammal {}
class Cow extends Mammal {}
class Horse extends Mammal {}
class ToyHorse extends Horse {}
```

and the following variables:

```
Object o; Mammal m; Cow c;  Horse h; ToyHorse t;
List<? extends Mammal> lem;
List<? extends Horse> leh;
List<? super Horse> lsh;
```

For each of the following, circle OK if the statement has the correct Java types and will compile, otherwise circle ERROR.

a) `lem.add(h);`	OK	ERROR
b) `lsh.add(t);`	OK	ERROR
c) `lsh.add(o);`	OK	ERROR
d) `lem.add(null);`	OK	ERROR
e) `h = lsh.get(1);`	OK	ERROR
f) `m = leh.get(1);`	OK	ERROR
g) `o = lsh.get(1);`	OK	ERROR

Question 5 (8 points)

Consider this implementation of a binary search. Draw the control-ow graph (CFG) for this implementation.

```
public static int binarySearch( int[] a, int val ) {
        int min = 0;
        int max = a.length - 1;
        while ( min < max ) {
                int mid = ( min + max ) / 2;
                if ( val == a[mid] ) {
                        min = mid;
                        max = mid;
                }
                else if ( val > a[mid] ) {
                        min = mid + 1;
                }
                else {
                        max = mid - 1;
                }
        }
        return max;
}
```

Question 6 (11 points)

Consider the following implementation of a stack data structure.
In a stack, elements are added to the top of the stack via push() and removed from the top of the stack via pop(). Note that there are a few bugs in the given implementation.

```
public class LongStack {
        private int maxSize;
        private long[] stackArray;
        private int top;

        public LongStack( int maxSize ) {
                this.maxSize = maxSize;
                this.stackArray = new long[maxSize];
                top = -1;
        }

        public void push( long j ) { stackArray[++top] = j; }
        public long pop() { return stackArray[top--]; }
        public boolean isEmpty() { return ( top == -1 ); }
        public boolean isFull() { return ( top == maxSize - 1 ); }
}
```

a) (5 points) Write a suitable representation invariant for the LongStack class.

b) (6 points) Make minimal corrections to the implementation of the LongStack class to fix any bugs. Note that you are only allowed to make changes to the method implementations (i.e., the representation fields and method signatures cannot be changed).

Rewrite methods in the space below, as necessary.

Question 7 (10 points)

Consider the Interval class shown below, which represents an interval of time between two Date objects (i.e., start and stop).

```
public class Interval {
        private Date start;
        private Date stop;
        private long duration;

        // Rep invariant: duration = stop.getTime() - start.getTime();

        public Interval( Date start, Date stop ) {
                this.start = start;
                this.stop = stop;
                duration = stop.getTime() - start.getTime();
        }

        public Date getStart() { return start; }
        public Date getStop() { return stop; }
        public long getDuration() { return duration; }
        public void setStart( Date start ) { this.start = start; }
        public void setStop( Date stop ) { this.stop = stop; }
}
```

List all ways in which the representation invariant does not hold.

Question 8 (10 points)

Assume the following code compiles:

```
    A a = new A();
    B b = new C(a);
    D d = b;
```

Mark all of the following that must be true:

a) A is a supertype of B.
b) A is a subtype of B.
c) B is a supertype of C.
d) B is a subtype of C.
e) B is a subtype of D.
f) B is a supertype of D.
g) C is a supertype of D.
h) C is a subtype of D.
i) A is a supertype of D.
j) A is a subtype of D.

Question 9 (10 points)

The following partial class definition is for an implementation of a polynomial with integer coefficients.

```
    class IntPoly {
        private int [] coeffs;  // the integer coefficients
        private int degree;   // the degree of the polynomial

        // the rest of the class definition follows….
    }
```

Write a suitable rep invariant for this class.

Next, write an abstraction function.

Question 10 (8 points)

```
class X {
     X() { }
}
class Y extends X {
     Y() { }
}
class Z {
     Z() { }
}
class W extends Z {
     W() { }
}
class A {
     A() {}
     X m(Z z) {
          System.out.println("A: X m(Z z)");
          return new X();
     }
}
class B extends A {
     B() { }
     X m(W w) {
          System.out.println("B: X m(W w)");
          return new X();
     }
}
class C extends B {
     C() { }
     Y m(W w) {
          System.out.println("C: Y m(W w)");
          return new Y();
     }
}
public class SubclassDemo3 {
     public static void main(String[] args) {
          A a = new C();
          W w = new W();
          // Which m is called?
          X x = a.m(w);

          C c = new C();
          W w2 = new W();
          // Which m is called?
          Y x3 = c.m(w2);
     }
}
```

What is printed when the code above is executed?

Question 11 (6 points)

The implementation of equals in class Object returns true if two references are exactly the same, i.e., a.equals(b) returns the result of the comparison a==b.

a) Show that this definition defines proper equivalence relationship. That is, show that it is reflexive, symmetric, and transitive.

b) The implementation of hashCode in Object returns the memory address of the object. Explain why this implementation of hashCode provides an effective hash function.

Consider the Property class shown below

```
public class Property {
     String description; // property description
     int weight; // weight

     public String getDescription() { return description; }
     public int getWeight() { return weight; }

     @Override
     public boolean equals(Object o) {
          if (! (o instanceof Property))
               return false;
          Property p = (Property) o;
          return this.description.equals(p.description) &&
                    this.weight == p.weight;
     }
/* .... */
}
```

Question 12. (6 points)
Here are six possible `hashCode` methods for `Property`. Your job is to decide which ones are legal (satisfy the contract for `hashCode`).

In the blank space to the left of each method, you should put an X if that `hashCode` is not a valid hashCode for class `Property`.

Note: `Math.random()` returns a real number in the range 0.0-1.0.

```
____ (i) int hashCode() { return description.hashCode(); }
____ (ii) int hashCode() { return 13*description.hashCode(); }
____ (iii) int hashCode() { return (int)(Math.random()*
                              description.hashCode()); }
____ (iv) int hashCode() { return 17; }
____ (v) int hashCode() { return weight; }
____ (vi) int hashCode() { return 13*description.hashCode() +
                         weight; }
```