Question 1 - Implementation of recursive data structures  Consider the following object-oriented implementation of lists of integers outlined in class (we omit for simplicity handling of error situations by exceptions).

```java
abstract class List {
    abstract boolean isEmpty ();
    abstract int head ();
    abstract List tail ();
}

class NonEmptyList extends List {
    private int head;
    private List tail;
    NonEmptyList (int head, List tail) {
        this.head = head; this.tail = tail;
    }
    boolean isEmpty () { return false; }
    int head () { return head; }
    List tail () { return tail; }
}
class EmptyList extends List {
    boolean isEmpty () { return true; }
    int head () { ... error }
    List tail () { ... error }
}
```

1. add in List non-abstract methods which implement shallow copy and the corresponding equality (denoted by (1) in course notes) for lists;
2. add in List non-abstract method which implement deep copy and the corresponding equality (denoted by (2) in course notes) for lists (assume that lists are acyclic);
3. add a heir class Interval as outlined in course notes, that is, whose instances represent intervals $[n, m]$ with $n \leq m$.
4. add a new observation isOrdered which tests whether a list is ordered (where? discuss different possibilities).

Question 2 - Wrappers for natural numbers  A natural number is either zero, or the successor of a natural number. Applying the methodology for implementing union types in object-oriented languages, natural numbers can then be implemented by defining an abstract class Nat with two heir classes Zero and Succ. Instances of these classes “represent” natural numbers.

1. Write these three classes with suitable fields and constructors. Then, add the following methods (abstract in Nat):
2. int val () returns the natural number represented by the receiver
3. Nat add (Nat n) returns a Nat object which represents the sum of (the naturals represented by) the receiver and the argument
4. Nat mult (Nat n) returns a Nat object which represents the product of (the naturals represented by) the receiver and the argument
5. boolean isZero () tests whether the receiver is (a representation of) zero;
6. boolean equals (Nat n) tests whether two Nat objects represent the same natural.
Question 3 - Dynamic binding, static binding, super

Consider the following Java classes:

class Parent {
    int k = 1;
    int m () { return 1;}
    int g () { return m(); }
    int km () { return k; }
}
class Heir1 extends Parent {
    int f () { return m();}
    int m () { return 2;}
}
class Heir2 extends Parent {
    int k = 2;
    int f () { return super.g(); }  
    int m () { return 3;}
    int km () { return k; }
}

public class Main {
    public static void main(String[] argv) {
        Parent p; Heir1 h1; Heir2 h2;
        p = new Heir1(); h2 = new Heir2();
        ...
    }
}

Write, for each of the cases, what happens at compile time and what happens, in case of successful compilation, at run-time, explaining why, if we replace dots with the code below.

1. System.out.println(p.f());
2. System.out.println(p.g());
3. h1 = p; System.out.println(p.m());
4. p = h2; System.out.println(p.m());
5. System.out.println(h2.f());
6. p = h2; System.out.println(p.k);
7. p = h2; System.out.println(p.km());