Question 1

What is dynamic binding? Which of the following does dynamic binding apply to in Java: (a) invocation of a static method; (b) invocation of an instance method; (c) access of a class variable; and (d) access of an instance variable?
Question 2: Program comprehension

2.1
Consider the function f() defined below:

```java
static void f(LinkedList<Integer> alist){
    ListIterator<Integer> iter;

    while(alist.size() != 0) {
        iter = alist.listIterator();
        for(int j = 0 ; j < alist.size()-1; j++)
            iter.next();
        System.out.println(iter.next());
        iter.remove();
    }
}
```

Assume `alist` has values [1,2,9,8,10]. What is the output of `f(alist)`?

2.2
Consider the following function f:

```java
static <T> void f(ArrayList<T> v) {
    int i, n;
    n = v.size();
    for (i = 1; i < n; i++){
        v.set(i, v.get(i-1));
    }
    v.remove(0);
    System.out.println(v);
}
```

Assume `v` has values [1,2,3,4,5]. What is the content of `v` after the function call `f(v)`?
2.3

Describe the behavior of the function `f` defined below.

```java
static <T> void f(T arr[]){
    int n = arr.length;
    LinkedList<T> s = new LinkedList<T>();
    int i;

    for (i = 0; i < n; i++)
        s.add(arr[i]);

    i = 0;
    while (!s.isEmpty()){  
        arr[i] = s.poll();
        i++;
    }
}
```

Assume `arr` contains values `[1, 5, 4, 3, 2]`. What is the content of `arr` after the function call `f(arr)`?

2.4

Does the following programs compile and run? If no, explain the reason. If yes, give the output.

```java
class A {
    public A(){
        System.out.println("A' constructor");
    }

    public void m(){
        System.out.println("A's m");
    }
}

class B extends A {
    public B(){
        System.out.println("B's constructor");
    }
    public void m(){
        System.out.println("B's m");
    }

    public static void main(String[] args){
        B o = new A();
        o.m();
    }
}
```
Question 3

Implement a class, named `SortedArrayList`, that extends the `java.util.ArrayList` with the following specification:

class SortedArrayList<E extends Comparable<E>> extends ArrayList<E> {
    public boolean add(E elm);
    public ArrayList<ArrayList<E>> sublists(int n);
}

- `SortedArrayList` overrides the `add` method in the following way: it inserts the object into the collection such that the collection remains sorted in ascending order after insertion. This operation should take linear time in the worst case.

- The class `SortedArrayList` provides a method, named `sublists(int n)`, that returns all `n`-element sublists of this list. For example, if this list is `["a","b","c","d"]` and `n` is 2, then the returned list is: `[ ["a","b"], ["b","c"], ["c","d"] ]`.

    public ArrayList<ArrayList<Object>> sublists(int n)
Question 4:

Consider the class ListNode:

class ListNode<E> {
    E data;
    ListNode<E> next;

    public ListNode(E data) {
        this.element = data;
        next = null;
    }

    public ListNode(E data, ListNode<E> next) {
        this.element = data;
        this.next = next;
    }
}

Write each of the following functions:

(a) static <E> boolean prefix(ListNode<E> preHead, ListNode<E> head)
    This function returns true if the list represented by preHead is a prefix of the list represented by head.

(b) static <E> boolean suffix(ListNode<E> sufHead, ListNode<E> head)
    This function returns true if the list represented by sufHead is a suffix of the list represented by head.
**Question 5**

Consider the `BTNode` class:

```java
class BTNode<T> {
    T data;
    BTNode<T> left, right;

    public BTNode(T data) {
        this.data = data;
        left = null;
        right = null;
    }
}
```

The following gives a function that tests if a tree is balanced:

```java
public static <E> boolean balanced(TreeNode<E> root) {
    if (root == null)
        return true;
    if (Math.abs(depth(root.left) - depth(root.right)) > 1) {
        return false;
    }
    return balanced(root.left) && balanced(root.right);
}
```

The used algorithm is not efficient since it visits a node multiple times. Give an implementation of a linear-time algorithm for the function that visits each node exactly once.
**Question 6 (extra-credit)**

The BC constraint, which occurs in many combinatorial problems, takes the following form: \( \sum_i (a_i \times b_i) \leq k \), where each \( a_i \) is a positive integer, each \( b_i \) is a 0/1 variable, and \( k \) is a positive integer. A valuation of the variables satisfies the constraint if the summation on the left-hand side does not exceed \( k \). For example, the constraint \( 2 \times b_1 + 3 \times b_2 + 5 \times b_3 \leq 5 \) has the following solutions: \([0,0,0], [0,0,1], [0,1,0], [1,0,0], [1,1,0]\). It can be checked that \([1,1,0]\) is a solution since \( 2 \times 1 + 3 \times 1 + 5 \times 0 \leq 5 \) is true. Write a function of the following specification that takes a BC constraint, and returns all the solutions:

```java
public static ArrayList<ArrayList<Integer>> solveBC(int[] a, int k)
```

For the above example, \( a = \{2,3,5\} \) and \( k = 5 \).