Pokud zadání nerozumíte nebo se vám zdá nejednoznačné, zeptejte se. Pište čitelně, nečitelná řešení nebudeme uznávat.

1. Odkrokujte následující program a s použitím notace z přednášky popište stav paměti v místech označených komentáři (stavy A, B a C). Pro B stačí stav poslední návštěvy příslušného místa v kódu. Předpokládejte, že garbage collector odstraní z haldy alokované instance okamžitě poté, co přestanou být z programu dostupné.

```
class Node {
    final int v;
    Node next;

    Node(int v, Node next) {
        this.v = v;
        this.next = next;
    }

    boolean f(int t) {
        // stav B
        return v == t || (next != null && next.f(t));
    }

    public static void main(String[] args) {
        Node n = new Node(2, new Node(3, new Node(1, null)));
        new Node(1, n);
        // stav A
        if (n.f(1)) {
            n.next = null;
        }
        // stav C
    }
}
```

2. Napište, co vypíše následující kód.

```
interface Alice {
    int v();
    Alice f(Alice a);
}


class Bob implements Alice {
    private final int v;
    public Bob(int v) { this.v = v; }
    public int v() { return v; }
    public Alice f(Alice a) { return new Bob(v + a.v()); }
}


class Cyril extends Bob {
    public Cyril(int v) { super(v); }
    public Alice f(Alice a) { return new Bob(v() - a.v()); }
}

class Dana extends Cyril {
    public Dana(int v) { super(v); }
    public Alice f(Alice a) {
        if (a.v() <= 0) return new Bob(1);
        return a.f(new Dana(v() - 1));
    }
}

public class Ex2 {
    public static void main(String[] args) {
        Bob b = new Bob(6);
        Cyril c = new Cyril(6);
        Dana d = new Dana(5);
        System.out.println(d.f(b).v());
        System.out.println(d.f(c).v());
    }
}
```

3. Najděte a vysvětlete chybu při překladu.

```
interface Question{

    public void ask();

}

abstract class Why implements Question{

    public void ask(){
       System.out.println("Why?");
    }

    abstract public void reply();
}

class DumbWhyReply extends Why{

    public void reply() {
        System.out.println("Why not.");
    }
}


class SmartWhyReply extends Why{

    public void reply() {
        System.out.println("I don't know.");
    }
}

class Interview{

    public static void main(String [] args){
        Question q1 = new DumbWhyReply();
        Why q2 = new SmartWhyReply();
        Question q3 = new DumbWhyReply();
        q1.ask();
        q1 = (Question) q2;
        q2.reply();
        q3 = (Why) q1;
        q3.ask();
        q1.reply();
    }
}
```

4. Vyznačte řádek, na kterém dojde k chybě za běhu programu. K jakému typu chyby dojde (nemusíte psát přesný název výjimky, stačí popsat typ chyby, např. chybné přetypování, přetečení zásobníku apod.)? Jak by se musel kód upravit, aby k chybě nedošlo? Jak bude vypadat nejdelší souvislý řetězec Node v době chyby?

```
class Node {
    public int value;
    private Node next;

    Node(int value) {
        this.value = value;
    }

    public boolean contains(int i){
        Node iterator = this;
        while (iterator.next != null){
            iterator = iterator.next;
            if(iterator.value == i) return true;
        }
        return false;
    }

    public void prepend(Node prev){
        prev.next = this;
    }

    public void append(Node newNode){
        Node iterator = next;
        if(next == null){ next = newNode; }
        while(iterator.next != null){
            iterator = iterator.next;
        }
        iterator.next = newNode;
    }
}


class Main{
    public static void main(String [] args){
        Node n1 = new Node(1);
        Node n2 = new Node(2);
        Node n3 = new Node(3);
        n1.append(new Node(4));
        n2.prepend(n3);
    }
}
```

5. V níže uvedeném kódu se zbavte podmínek (tj. příkazů if a switch). Pokud budete potřebovat můžete si dodefinovat nové pomocné metody a třídy. Rozhraní třídy Dragon musí zůstat zachováno.

```
class Princess {

}

class Dragon {

    protected final static int HUNGRY = 1;
    protected final static int SLEEPING = 2;
    protected final static int COOL = 3;
    protected int mood = HUNGRY;

    public void meets(Princess p) throws Exception {
        switch (mood) {
            case HUNGRY:
                this.eat(p);
                mood = SLEEPING;
                break;
            case SLEEPING:
                mood = COOL;
                break;
            case COOL:
                this.say(p, "Hey, how are you?");
                mood = HUNGRY;
                break;
            default:
                throw new Exception("Invalid mood!");
        }
    }

    public void eat(Princess p) {
        /* ... */
    }

    public void say(Princess to, String what) {
        /* ... */
    }
}
```