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 sledujte stav paměti v místech označených komentáři (stavy A, B a C). Napište pouze pro **první** návštěvu příslušného místa. 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(a.v() - 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()/10));
    }
}

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.

```
abstract class Animal {
    public double skill;

     public Animal() {
         skill = Math.random();
     }

     public abstract void catchAnimal(Animal animal);

     public void escapeFrom(Dog dog) {
         skill = (skill >= dog.skill) ? skill + dog.skill : skill - dog.skill;
     }

    public void escapeFrom(Cat cat) {}
}

 class Dog extends Animal {
     public void catchAnimal(Animal animal){
           animal.escapeFrom(this);
     }

     public void escapeFrom(Cat cat) {
         skill++;
     }

     public static Animal compare(Dog a, Dog b){
             return (a.skill >= b.skill) ? a : b;
     }
 }

 class Cat extends Animal {
     public void catchAnimal(Animal animal){
         animal.escapeFrom(this);
     }
 }

class Combat{
    public static void main(String [] args) {
        Animal c1 = new Cat();
        Dog d1 = new Dog();
        Animal d2 = new Dog();
        d2.catchAnimal(c1);
        d2 = (Dog) d1;
        d1.catchAnimal(c1);
        d2 = Dog.compare(d2, d1);
        d2.catchAnimal(d1);
    }
 }
```

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 vytvořený strom?

```
class Node{
    int value;
    Node left, right;

    public Node(int i){
        value = i;
        left = (i > 0) ? new Node(i-1) : null;
    }

    public boolean contains(int i){
        if (value == i) return true;
        return right.contains(i) || left.contains(i);
    }

    public void insert(Node node){
        if(left == null || left.value < node.value) {
            node.left = left;
            this.left = node;
        }else{
            node.right = right;
            this.right = node;
        }
    }
}

class Main{
    public static void main(String args[]){
        Node n1 = new Node(2);
        Node n2 = new Node(3);
        n1.contains(2);
        n1.insert(n2);
        n1.contains(3);
    }
}
```

5. V níže uvedeném kódu odstraňte aktivní čekání. Pokud budete potřebovat můžete si dodefinovat nové pomocné metody a třídy. Rozhraní tříd Worker a Machine musí zůstat zachováno.

```
class Machine {

    boolean isReady = true;

    public void doSomeStuff() throws InterruptedException {
        isReady = false;
        // A lot of work.....
        // invoke asyncMethod
        Thread t = (new Thread() {
            @Override
            public void run() {
                // do stuff
            }
        });
        t.start();
        t.join();
    }

    public void asyncMethod() {
        // do a lot of work
        // work is done
        isReady = true;
    }
}

class Worker {

    Machine m;

    public Worker(Machine m) {
        this.m = m;
    }

    public void doWork() {
            // watch isReady
            while (!m.isReady) {
                    Thread.sleep(100);
            }
            m.doSomeStuff();
    }
}

public class Factory {
     public static void main(String[] args) {
         Machine m = new Machine();
         Worker w;
```

```
        for(int i = 0; i<10; ++i) {
            w = new Worker(m);
            w.doWork();
        }
    }
}
```