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
    }
}
```

Řešení:

Následující seznam ukazuje stav paměti, na místech označených komentáři.

stav A:

$$\left(\begin{array}{l|l|} \#1 \to Node(v=1, next=null) & \\ \#2 \to Node(v=3, next=\#1) & \\ \#3 \to Node(v=2, next=\#2) & n=\#3 \end{array} \right)$$

stav B: (první)

$$\left(\begin{array}{l|l|} \#1 \to Node(v=1, next=null) & this=\#3, t=1 \\ \#2 \to Node(v=3, next=\#1) & \hline \\ \#3 \to Node(v=2, next=\#2) & n=\#3 \end{array} \right)$$

stav C:

$$\left(\begin{array}{l|l|} \#3 \to Node(v=2, next=null) & n=\#3 \end{array} \right)$$

2. Napište, co vypíše následující kód.

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

class B implements A {
    private final int v;
    public B(int v) { this.v = v; }
    public int v() { return v; }
    public A f(A a) { return new B(a.v() - v); }
}

class C extends B {
    public C(int v) { super(v); }
    public A f(A a) { return new B(v() * a.v()); }
}

class D extends C {
    public D(int v) { super(v); }
    public A f(A a) {
        if (a.v() <= 0) return new B(1);
        return a.f(new D(v()/10));
    }
}

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

Poznámka: oproti původnímu zadání jsou zde zkrácena jména rozhraní a tříd.

Kód vypíše na samostatné řádky čísla -6 a 0. Číslo -6 získáme voláním `d.f(b).v()`, viz stavy paměti (vždy bezprostředně po volání příslušných metod):

stav D.f():

$$\left(\begin{array}{l|l} \#1 \to B(v=6) & this = \#3, a = \#1 \\ \#2 \to C(v=6) & \hline \\ \#3 \to D(v=5) & b = \#1, c = \#2, d = \#3 \end{array} \right)$$

stav B.f():

$$\left(\begin{array}{l|l} \#1 \to B(v=6) & this = \#1, a = \#4 \\ \#2 \to C(v=6) & \hline \\ \#3 \to D(v=5) & this = \#3, a = \#1 \\ \#4 \to D(v=0) & \hline \\ & b = \#1, c = \#2, d = \#3 \end{array}\right)$$

V případě čísla 0:

stav D.f():

$$\left(\begin{array}{l|l} \#1 \to B(v=6) & this = \#3, a = \#2 \\ \#2 \to C(v=6) & \hline \\ \#3 \to D(v=5) & b = \#1, c = \#2, d = \#3 \end{array}\right)$$

stav C.f():

$$\left(\begin{array}{l|l} \#1 \to B(v=6) & this = \#2, a = \#6 \\ \#2 \to C(v=6) & \hline \\ \#3 \to D(v=5) & this = \#3, a = \#2 \\ \#6 \to D(v=0) & \hline \\ & b = \#1, c = \#2, d = \#3 \end{array}\right)$$

V obou případech se uplatnilo celočíselné dělení $6/10 = 0$.

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);
    }
 }
```

K problému při překladu dojde při porovnávání *d2 = Dog.compare(d2, d1);*. Protože metoda očekává oba vstupy typu *Dog*, zde je však *d2* typu *Animal*.

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);
    }
}
```

K chybě dojde při vyhodnocování volání *n1.contains(3);*, konkrétně v metodě *contains* při vyhodnocování *right.contains(i)*, protože *right* je *null*. Výjimka tedy je *NullPointerException*.

Návrh na opravu:

```
 public boolean contains(int i){
        if (value == i) return true;
        if (right == null && left == null) return false;
        if (right == null && left != null) return left.contains(i);
        if (right != null && left == null) return right.contains(i);
        return right.contains(i) || left.contains(i);
```

```
  }
```

akceptované řešení bylo i méně odolné:

```
 public boolean contains(int i){
       return left.contains(i) || right.contains(i);
  }
```

nebo naznačení řešení pomocí vzoru NullObject.

Nejdelší strom vypadá následovně: $Node(2) \rightarrow Node(3) \rightarrow Node(1) \rightarrow\rightarrow Node(0)$, kde '$\rightarrow$' reprezentuje ukazatel *Node.left*. *Node.right* vždy ukazuje na *null*.

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();
         }
     }
}
```

Řešení: Zde se nabízelo použít Observer, kde třída Machine bude implementovat rozhraní Observable a třída worker rozhraní Observer.