E1. (příklad na spojové seznamy) Doprogramujte kód metody `extractSmallest` tak, aby ze spojového seznamu odkazovaného proměnnou `head` odstranila nejmenší prvek a vrátila jeho hodnotu. Dejte si pozor na krajní případy.

```
class Node {
    Node n;
    int c;

    static Node head;

    static int extractSmallest() {
        /* kod */
    }
}
```

```
class Node {
Node next;
int c;
static Node head;
static int extractSmallest() {
int min = 0;
Node predV = head;
if (head.next == null) {
    min = head.c;
    head = null; } // if

for (Node em = head; em.next != null; em = em.next) {
  if (em.next.c < predV.next.c) {
    predV = em; } // if } // for
  if (head.c < predV.next.c) {
    min = head.c; }
    else {min = predV.next.c; }
    return min; }
```

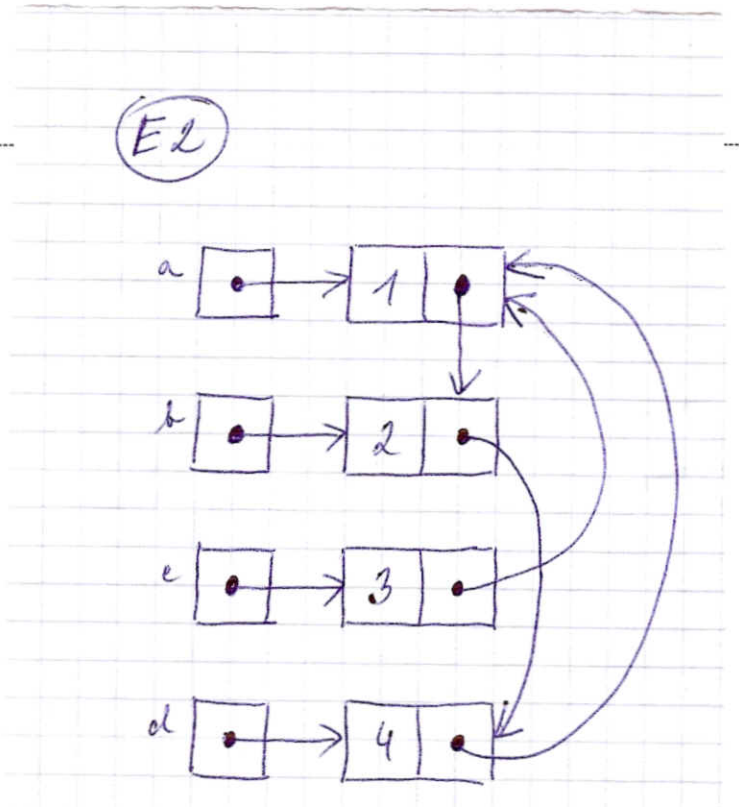
3.6. /1

E1 - verze II.:

```
static int extractSmallest() {
/* je-li head null nejsou data */
      if ( head == null ) return 0;

/* m ukazuje na prvek Node, ktery ma nejmensi hodnotu */ Node m = head;  // prvotni
inicializace, ukazuje na zacatek seznamu

/* projedeme cely ceznam, urcite tam bude nejmensi hodnota */ for ( Node t = head; t !=
null; t = t.n ) {

/* je-li hodnota v mensi zapamatuj si ji do m, je to nova nejmensi hodnota */ if ( m.c >
t.c ) { m=t;}// if }// for

/* Zde jiz m ukazuje na nejmensi hodnotu */ int result = m.c;   /* zapamatuj si hodnotu
cislo */

/* smazeme prvek tak, ze hodnotu "c" kam ukazuje head presuneme na pozici "m", to ukazoval
na prednasce je to nejjednodussi metoda, jinak se do toho clovek zamota a toto umoznuje i
docela elegantni osetreni okrajovych podminek */ m.c = head.c;  /* presun hodnotu */ head
= head.n;  /* preskoc prvni prvek, bude li jeden bude head ukazovat na null */

return result; }
```

E2. (příklad na spojové seznamy) Nakreslete, jak vypadají objekty v paměti po vykonání metody `f`.

```
class Node {
    Node n;
    int c;

    Node(int cont) { c = cont; }

    static void f() {
        Node a = new Node(1);
        Node b = new Node(2);
        Node c = new Node(3);
        Node d = new Node(4);
        a.n = b
        b.n = a;
        a.n.n = c;
        c.n = a;
        b.n = d;
        b.n.n = a;
    }
}
```



3.6. /2

E3. Metoda `top` ve třídě `ExtendedStack` vypisuje logovací hlášky o vybrání a opětovném vložení prvku, ale to se vám nelíbí. Zařiďte to tak, aby žádnou hlášku nevypisovala, vyhněte se duplikaci kódu.

```
class Stack {
    /* atributy */

    /** Vlozi prvek. */
    void push(int e) { /* kod */ }

    /** Vyjme a vrati posledni vlozeny prvek. */
    int pop() { /* kod */ }

    /** Vrati hodnotu posledniho vlozeneho prvku
        (bez vyjmuti). */
    int top() {
        int result = pop();
        push(result);
        return result;
    }
}

class ExtendedStack extends Stack {
    void push(int e) {
        System.out.println("Prvek vlozen.");
        super.push(e);
    }

    int pop() {
        System.out.println("Prvek vybran.");
        return super.pop();
    }
}
```

Ve tride ExtendedStack se prekryje metoda top timto kodem:

```
int top() {
      int result = super.pop();
      super.push(result);
      return result; }
```

Prida se tam super.xxx(), tim se vola metoda predka, ktera nevypisuje informace.

E4. Rozhraní `Animal` nemá žádné metody, takže třídy `Dog` a `Cat` nemusí mít žádné společné metody. Je takové rozhraní k něčemu? Pokud ano, uveďte příklad.

```
interface Animal {}
class Dog implements Animal { /* ... */ }
class Cat implements Animal { /* ... */ }
```

Ano, smysl má.
Dá se např. vytvořit pole
Animal a do něj ukládat
odkazy na objekty třídy
Cat a Dog.

(7K)

```
Animal [] animal = new Animal [3];
Cat cat = new Cat();
animal [1] = cat();
```

3.6. /3

E5. Je v následujícím kódu něco špatně (kromě chybějících těl metod)? Pokud ano, co?

```
interface Stack {
    void push(int e);
    int pop();
}

class StackImpl implements Stack {
    /* atributy */

    public void push(int e) { /* kod */ }
    public int pop() { /* kod */ }
    int top() { /* kod */ }

    static void f() {
        Stack s = new StackImpl();
        s.push(7);
        s.pop();
    }
}
```

Je to OK.

3.6. /4