E1. (příklad na spojové seznamy) Doprogramujte kód metody `add` tak, aby vkládala svůj parametr do spojového seznamu, na který ukazuje proměnná `head`. Výsledný seznam by měl být vzestupně seřazen podle obsahu instančních proměnných `c`. Dejte si pozor na krajní případy.

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

    static Node head;

    static void add(Node n) {
        /* kod */
    }
}
```

```
class Node {
Node next;
int c;
static Node head;

static void add (Node n) {
  n.next = null;
  if (head == null) { Node.head = n;
                      return; } //if

  Node vklad;
  for (Node em = head; em.next != null; em = em.next) {
    if (em.next == null) {
      em.next = n;
      return; } //if
    if (em.next.c > c) {
      vklad = em;
      break; } // if } // for
  n.next = vklad.next;
  vklad.next = n;
  } // add } // class
```

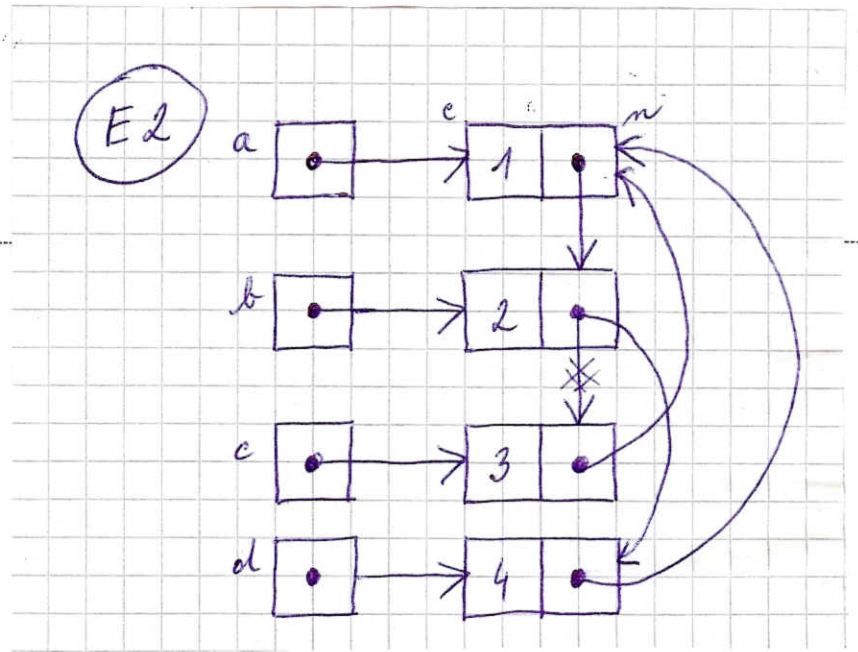
24. 5. / 1

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;
        a.n.n = c;
        c.n = a;
        b.n = d;
        b.n.n = a;
    }
}
```



E3. Ve třídě `Queue` je něco špatně. Napište co, proč a opravte to.

```
class Queue {
    int[] contents;
    int first, last;

    /** Metoda init musi byt zavolana
        pred prvnim pouzitim teto instance. */
    void init() { contents = new int[10000]; }

    /** Vlozi parametr dovnitr fronty. */
    void insert(int c) { contents[last++] = c; }

    /** Vrati nejstarsi prvek ve fronte. */
    int remove() { return contents[++first]; }
}
```

správně má být

```
int remove () {
    return contents [first ++]; }
```

DŮVOD:

++first = nejprve se „zvětší" o 1 a pak vypíše hodnotu

first ++ = nejprve vypíše hodnotu a pak se „zvětší" o 1

27.5. /2

E4. Zredukujte kód třídy `RedBlueSet`, zachovejte funkčnost metod `addRed`, `containsRed`, `removeRed`, `addBlue`, `containsBlue` a `removeBlue`. Pokud chcete, můžete část funkčnosti třídy `RedBlueSet` přesunout do jiné třídy (to je nápověda).

```
class RedBlueSet {
    int[] redContents;
    int redCount;
    int[] blueContents;
    int blueCount;

    void addRed(int c) {
        redContents[redCount++] = c;
    }

    int indexOfRed(int c) {
        for (int i = 0; i < redCount; i++)
            if (redContents[i] == c) return i;
        return -1;
    }

    boolean containsRed(int c) {
        return indexOfRed(c) != -1;
    }

    void removeRed(int c) {
        int i = indexOfRed(c);
        if (i == -1) return;
        redContents[i] = redContents[--redCount];
    }

    void addBlue(int c) {
        if (containsBlue(c)) return;
        blueContents[blueCount++] = c;
    }

    int indexOfBlue(int c) {
        for (int i = 0; i < blueCount; i++)
            if (blueContents[i] == c) return i;
        return -1;
    }

    boolean containsBlue(int c) {
        return indexOfBlue(c) != -1;
    }

    void removeBlue(int c) {
        int i = indexOfBlue(c);
        if (i == -1) return;
        blueContents[i] = blueContents[--blueCount];
    }
}
```

```
class RedBlueSet {
  MySet reds;
  MySet blues;

  void addBlue(int c) {
    blues.add(c);
  }
```

24.5. /3

E5. Vytvořte alternativní implementaci třídy `ExtendedStack`, která se chová stejně, ale nepoužívá dědičnost. Vyvarujte se duplikace kódu třídy `Stack`. Svoji volbu řešení podrobně zdůvodněte.

```
class Stack {
    int[] contents = new int[1000];
    int offset = 0;

    void push(int number) { contents[offset++] = number;}
    int pop() { return contents[--offset]; }
}

class ExtendedStack extends Stack {
    int top() { return contents[offset]; }
}
```

27.5. /4