E1. (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 = c;
        b.n.n = a;
        a.n = d;
        c.n.n.n = b;
    }
}
```

E2. (příklad na spojové seznamy) Doplňte kód metody `insertMaxToHead`, která projde spojový seznam, na jehož první prvek je odkazováno proměnnou `head`, a na jeho začátek vloží kopii maximální hodnoty nalezené v seznamu. Pro jednoduchost předpokládejte, že v seznamu jsou jen nezáporná celá čísla. Pozor na krajní případy. Pokud by byl seznam prázdný, vložte nulu.

```
class Node {
    static Node head;

    int value;
    Node next;

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

    public static void print() {  /* Vypíše seznam */ }
    public static void append (int value){ /* Přidá prvek
na konec seznamu */ }

    public static void insertMaxToHead(){ /* DOPLŇTE */ }

    public static void main (String[] args){
        insertMaxToHead();
        print(); /* vytiskne: 0 */
        append(3);
        append(1);
        insertMaxToHead();
        print(); /* vytiskne: 3,0,3,1 */
        append(4);
        append(5);
        insertMaxToHead();
        print(); /* vytiskne: 5,3,0,3,1,4,5 */
    }
}
```

E3. Rozhraní `Stack` předepisuje operace zásobníku. Odstraňte duplikaci kódu v třídách `ListStack` a `ArrayStack`, kromě modifikace rozhraní `Stack` můžete dělat libovolné úpravy.

```
interface Stack {
    void push (int value); /** vlozi prvek na vrchol **/
    int pop ();            /** vyjme prvek z vrcholu **/
    /** postupne vlozi prvky pole na zasobnik **/
    void multiPush(int[] values);
}

class ListStack implements Stack {
    static class StackNode {
        StackNode next;
        int value;
    };

    StackNode top;

    public void push (int value) {
        StackNode tmp = new StackNode();
        tmp.value = value;
        tmp.next = top;
        top = tmp;
    }

    public int pop () {
        int tmp = top.value;
        top = top.next;
        return tmp;
    }

    public void multiPush(int[] values){
        for (int i: values){
            push(i);
        }
    }
}

class ArrayStack implements Stack {
    int [] content = new int [10000];
    int top = 0;

    public void push (int value){
        content[top++] = value;
    }

    public int pop (){
        return content[--top];
    }

    public void multiPush(int[] values){
```

```
        for (int i: values){
            content[top++] = i;
        }
    }
}
```

E4. Co vypíše metoda `main`?

```
class Elem {

    int value = 0;

    public static void f(Elem a) {
        a.value = 3;
        g(a.value);
        a.value = a.value + 4;
    }

    public static void g(int a) {
        a = a + 3;
    }

    public static void main(String[] args) {
        Elem a = new Elem();
        f(a);
        System.out.println(a.value);
    }
}
```

E5. Následující kód lze sice zkompilovat a spustit, ale něco je na něm špatně. Co a proč?

```
import java.util.ArrayList;

class Pytel {
    ArrayList obsah = new ArrayList<Integer>();

    /* Prida prvek do pytle */
    void pridej(int prvek) {
        obsah.add(prvek);
    }

    /* Odebere prvek dane hodnoty z pytle */
    void odeber(int prvek){
        if (obsah.contains(prvek)){
            obsah.remove(new Integer(prvek));
        }
        else {
            throw new RuntimeException("NEMAM!!!");
        }
    }
}


class DeravyPytel extends Pytel {

    public static final int LIMIT = 300;

    /**
     * Prida prvek jen pokud je dostatecne velky
     * (nepropadne dirami v pytli ven)
     */

    void pridej(int prvek) {
        if (prvek > LIMIT) super.pridej(prvek);
    }
}
```

D1. Programátor se rozhodl naimplementovat třídu `List` (reprezentující seznam čísel) jako spojový seznam polí o maximálně deseti prvcích. Dopište do třídy `List` podporu návrhového vzoru iterátor.

```
class List {
    Chunk first;

    /* kod */
}

class Chunk {
    int[] contents = new int[10];
    int length;
    Chunk next;

    /* kod */
}
```

D2. Třída `List` opět reprezentuje seznam čísel. Upravte kód níže tak, aby šlo ve spojovém seznamu kromě instancí třídy `Chunk` libovolně střídat také instance tříd `Node` a `Pair` (upravit budete muset všechny třídy, neřešte změny v kódu, který nevidíte).

```
class List {
    Chunk first;

    /* kod */
}

class Chunk {
    int[] contents = new int[10];
    int length;
    Chunk next;

    /* kod */
}

class Node {
    int contents;
    Node next;

    /* kod */
}

class Pair {
    int first;
    int second;
    Pair next;

    /* kod */
}
```

D3. Do třídy `Cell` dopište podporu návrhového vzoru singleton. Dejte si pozor, aby tato podpora skutečně zajistila, že nepůjde vytvořit víc než jedna instance `Cell`.

```
class Cell {
    int contents;

    void set(int c) { contents = c; }
    int get() { return contents; }
}
```

D4. Proč je v níže uvedeném kódu `assert`? Jakou má funkci?

```
static int compare(int a, int b) {
    if (a < b) return -1;
    if (a > b) return 1;
    assert false;
    return 1;
}
```

D5. Do třídy `Cell` z příkladu D3 dopište podporu návrhového vzoru observer. Dejte si pozor, abyste událost rozesílali pouze tehdy, když se hodnota uložená v dané instanci skutečně změní.

C1. Navrhněte strukturu tříd (tzn. rozhraní, třídy, jejich instanční proměnné a hlavičky metod) odpovídající návrhovému vzoru interpretr, která by byla schopná interpretovat následující kód:

```
begin
  a = 8
  b = 9
  r = 0
  while a > 0 do begin
    r = r + b
    a = a – 1
  end
end
```

C2. Napište kód implementující návrhový vzor abstract factory: továrna by měla být schopná vracet `BlueCircle` a `BlueSquare` nebo `RedCircle` a `RedSquare` v závislosti na tom, jestli je ve stavu *blue* nebo *red*.

C3. Java za `assert` dovolí napsat libovolný kód vracející hodnotu typu boolean. To však neznamená, že tam libovolný kód skutečně může být – čeho by se měl kód za `assert` vyvarovat? Uveďte příklad.

C4. Změňte níže uvedený kód tak, abyste se zbavili použití operátoru `instanceof`. Pokud chcete, můžete změnit také design souvisejícího kódu. Inspirujte se návrhovým vzorem visitor, vyhněte se řešením založeným na metodách typu `getTypeId`.

```
void f(Vehicle v) {
    if (v instanceof Car) v.g();
    else v.h();
}
```

C5. Na příkladu popište funkci návrhového vzoru stav.