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 popište stav paměti v místech označených komentáři (stavy A, B a C). Předpokládejte, že typ `double` má neomezenou přesnost a že garbage collector odstraní z haldy alokované instance okamžitě poté, co přestanou být z programu dostupné.

```
class Vector {
    final double x, y;
    private static Vector testVector = null;

    Vector(double x, double y) {
        this.x = x;
        this.y = y;
        // stav B
    }

    double getX() {
        return x;
    }

    double getY() {
        return y;
    }

    static Vector testVector() {
        // stav A
        if (testVector == null) testVector = new Vector(1.0, 0.0);
        return testVector;
    }

    Vector funcionMagica() {
        double[] a = {1.0, 0.5, -1.0, -1.0};
        Vector v = this;
        for (double d : a) v = new Vector(Math.sin(Math.PI * d),
                    Math.cos(Math.PI * d));
        return v;
    }

    public static void main(String[] args) {
        Vector v = Vector.testVector();
        double k = v.getY() / v.getX();
        Vector v2 = v.funcionMagica();
        // stav C
    }
}
```

Řešení:
Následující seznam ukazuje stav paměti, na místech označených komentáři.

Pozn: protože jsme si nedefinovali notaci pro pole, která se stejně jako objekty ukládají na heap, je jako cílová adresa zapsáno "NA"(not available). Pokud náhodou ve vašem zadání narazíte na pole, klidně jej úplně ignorujte, případně se zeptejte.

stav A:

$$\left(\;\middle|\; \text{———————} \;\middle|\; Vector.testVector = null \;\right)$$

stav B:

$$\left(\begin{array}{l|l|l} \#1 \to Vector(x=1,y=0) & this = \#1, x = 1, y = 0 & Vector.testVector = null \\ & \text{———————} & \\ & \text{———————} & \end{array} \right)$$

stav B:

$$\left(\begin{array}{l|l|l} \#1 \to Vector(x=1,y=0) & this = \#2, x = 0, y = -1 & Vector.testVector = \#1 \\ \#2 \to Vector(x=0,y=-1) & \text{———————} & \\ & this = \#1, a = NA, v = \#1, d = 1 & \\ & \text{———————} & \\ & v = \#1, k = 0 & \end{array} \right)$$

stav B:

$$\left(\begin{array}{l|l|l} \#1 \to Vector(x=1,y=0) & this = \#3, x = 1, y = 0 & Vector.testVector = \#1 \\ \#2 \to Vector(x=0,y=-1) & \text{———————} & \\ \#3 \to Vector(x=1,y=0) & this = \#1, a = NA, v = \#2, d = 0.5 & \\ & \text{———————} & \\ & v = \#1, k = 0 & \end{array} \right)$$

stav B:

$$\left(\begin{array}{l|l|l} \#1 \to Vector(x=1,y=0) & this = \#4, x = -0, y = -1 & Vector.testVector = \#1 \\ \#3 \to Vector(x=1,y=0) & \text{———————} & \\ \#4 \to Vector(x=-0,y=-1) & this = \#1, a = NA, v = \#3, d = -1 & \\ & \text{———————} & \\ & v = \#1, k = 0 & \end{array} \right)$$

stav B:

$$\left(\begin{array}{l|l|l} \#1 \to Vector(x=1,y=0) & this = \#5, x = -0, y = -1 & Vector.testVector = \#1 \\ \#4 \to Vector(x=-0,y=-1) & \text{———————} & \\ \#5 \to Vector(x=-0,y=-1) & this = \#1, a = NA, v = \#4, d = -1 & \\ & \text{———————} & \\ & v = \#1, k = 0 & \end{array} \right)$$

stav C:

$$\left(\begin{array}{l|l|l} \#1 \to Vector(x=1,y=0) & & Vector.testVector = \#1 \\ \#5 \to Vector(x=-0,y=-1) & v = \#1, k = 0, v2 = \#5 & \end{array} \right)$$

2. Napište, co vypíše následující kód.

```
abstract class A {
    abstract void fa(A a);

    void fb(B b) {
        System.out.println("1");
    }

    void fc(C c) {
        System.out.println("2");
    }
}

class B extends A {
    void fa(A a) {
        a.fb(this);
    }

    void fc(C c) {
        System.out.println("3");
    }
}

class C extends A {
    void fa(A a) {
        a.fc(this);
    }

    void fb(B b) {
        System.out.println("4");
    }
}

class Main {
    public static void main(String args[]) {
        new B().fa(new B());
        new B().fa(new C());
        new C().fa(new C());
        new C().fa(new B());
    }
}
```

Kód vypíše na samostatné řádky čísla: 1, 4, 2, 3.

3. Najděte a vysvětlete chybu při překladu.

```
class A {
    static int i = 0;

    static void f() {
        System.out.println("A.f()");
    }

    void g() {
        System.out.println(h());
    }

    private String h() {
        return "A.h()";
    }
}

class B extends A {
    static void f() {
        System.out.println("B.f()");
        g();
    }

    private String h() {
        return "B.h(): " + (++i);
    }
}

class Task3 {
    public static void main(String[] args) {
        A a = new B();
        a.f();
        a.g();
    }
}
```

Statická metoda `B.f()` se odkazuje na nestatickou metodu `g()`.

4. Vyznačte řádek, na kterém dojde k chybě za běhu programu, a napište na které instanci k dané chybě došlo. Svoji odpověď vysvětlete.

```
class Main {
    public static void main(String args[]) {
        A a = new A();
        B b1 = new B(a);
        B b2 = new B(b1);

        System.out.println(b2.f());
    }
}

interface I {
    int f();

    void g();
}

class A implements I {
    I a;
    int c;

    public int f() {
        return a.f() + c;
    }

    public void g() {
        this.a = new B(a);
        c = 70;
    }
}

class B extends A {
    public B(I t) {
        a = t;
        c = 30;
    }

    public int f() {
        c += 30;
        return c > 50 ? super.f() + c : c + 50;
    }
}
```

Před zavoláním `B.f()` na instanci `b2`[1] vypadá stav paměti následovně:

$$\left(\begin{array}{l|l} \#1 \to A(a = null, c = 0) & \\ \#2 \to B(a = \#1, c = 30) & \\ \#3 \to B(a = \#2, c = 30) & a = \#1, b1 = \#2, b2 = \#3 \end{array}\right)$$

Dále je metoda `B.f()` zavolána (stav ihned po jejím zavolání):

$$\left(\begin{array}{l|l} \#1 \to A(a = null, c = 0) & this = \#3 \\ \#2 \to B(a = \#1, c = 30) & \hline \\ \#3 \to B(a = \#2, c = 30) & a = \#1, b1 = \#2, b2 = \#3 \end{array}\right)$$

Metoda zvýší hodnotu `b2.c` o 30 a protože podmínka $c(= 60) > 50$ je splněna, volá na stejné instanci (`b2`) metodu `A.f()`:

$$\left(\begin{array}{l|l} \#1 \to A(a = null, c = 0) & this = \#3 \\ \#2 \to B(a = \#1, c = 30) & \hline \\ \#3 \to B(a = \#2, c = 60) & this = \#3 \\ & \hline \\ & a = \#1, b1 = \#2, b2 = \#3 \end{array}\right)$$

Metoda obratem volá metodu `B.f()` pro instanci `b1`:

$$\left(\begin{array}{l|l} \#1 \to A(a = null, c = 0) & this = \#2 \\ \#2 \to B(a = \#1, c = 30) & \hline \\ \#3 \to B(a = \#2, c = 60) & this = \#3 \\ & \hline \\ & this = \#3 \\ & \hline \\ & a = \#1, b1 = \#2, b2 = \#3 \end{array}\right)$$

Metoda zvýší hodnotu `b1.c` o 30 a protože podmínka $c(= 60) > 50$ je, stejně jako v minulém případě splněna, volá na stejné instanci (`b1`) metodu `A.f()`:

$$\left(\begin{array}{l|l} \#1 \to A(a = null, c = 0) & this = \#2 \\ \#2 \to B(a = \#1, c = 60) & \hline \\ \#3 \to B(a = \#2, c = 60) & this = \#2 \\ & \hline \\ & this = \#3 \\ & \hline \\ & this = \#3 \\ & \hline \\ & a = \#1, b1 = \#2, b2 = \#3 \end{array}\right)$$

Metoda `A.f()` obratem volá metodu `A.f()` pro instanci `a`:

[1] Správně bychom měli použít pro specifikaci instance její adresu #3. Zápis `b2` je však srozumitelnější.

$$\left(\begin{array}{l|l|l} \#1 \to A(a = null, c = 0) & this = \#1 & \\ \#2 \to B(a = \#1, c = 60) & \hline this = \#2 & \\ \#3 \to B(a = \#2, c = 60) & \hline this = \#2 & \\ & \hline this = \#3 & \\ & \hline this = \#3 & \\ & \hline a = \#1, b1 = \#2, b2 = \#3 & \end{array}\right)$$

Metoda `A.f()` se pokusí zavolat metodu `f()` pro instanci na adrese odkazované atributem `a.a`. Ten je však nastaven na `null`, proto program havaruje s chybou `NullPointerException` na řádku `"return a.f() + c"` metody `A.f()`. Pozn: pokud to není v zadání výslovně napsáno, nemusíte notaci používat.

5. Třída `Table` reprezentuje tabulku, ve které může každé číslo být asociováno se žádným, jedním nebo více řetězci. Metoda `add(Integer key, String value)` přidá do tabulky asociaci klíče `key` s hodnotou `value`, metoda `remove(Integer key, String value)` odstraní z tabulky asociaci klíče `key` s hodnotou `value` a metoda `getValues(Integer key)` vrátí množinu všech hodnot asociovaných s klíčem `key`. Vaším úkolem je dopsat kód metody `add`, kód ostatních metod ani atributy modifikovat nesmíte.

```
class Table {
    private Map<Integer, Set<String>> map = new HashMap<>();

    public void remove(Integer key, String value) {
        if (map.containsKey(key)) {
            Set<String> set = map.get(key);
            set.remove(value);
            if (set.isEmpty()) map.remove(key);
        }
    }

    public Set<String> getValues(Integer key) {
        Set<String> set = map.get(key);
        if (set == null) set = new HashSet<>();
        return set;
    }

    public void add(Integer key, String value) {
        Set<String> set = map.get(key);
        if (set == null) set = new HashSet<>();
        set.add(value);
        map.put(key, set); //nahrazuje položku s existujícím klíčem
    }
}
```