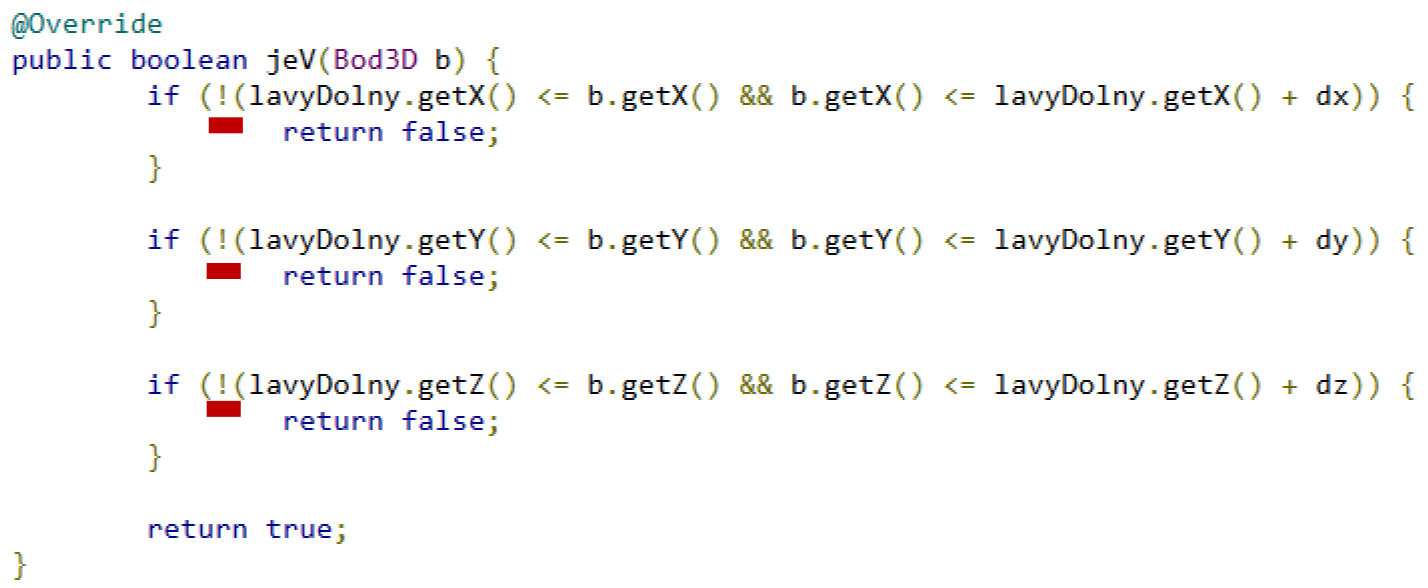
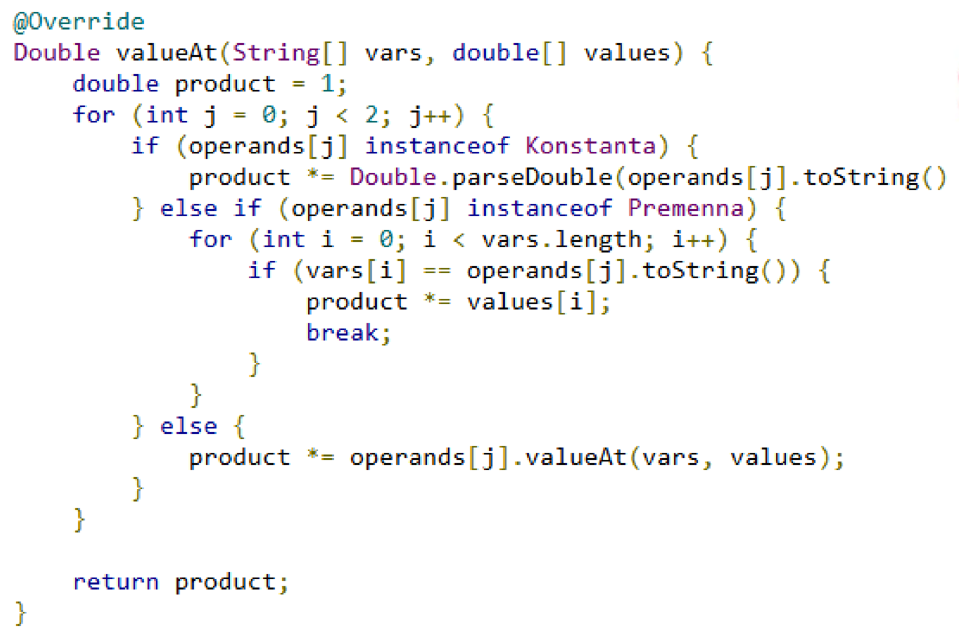
Podmienky sú čitateľnejšie v pozitívnom smere bez použitia negácie na začiatku.



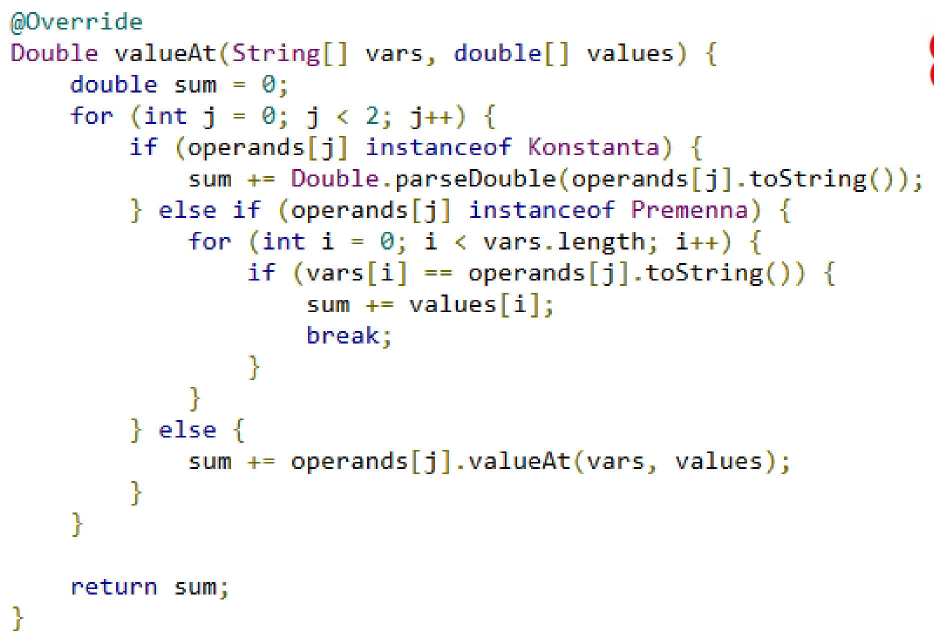
```
@Override
public boolean jeV(Bod3D b) {
        if (!(lavyDolny.getX() <= b.getX() && b.getX() <= lavyDolny.getX() + dx)) {
                return false;
        }

        if (!(lavyDolny.getY() <= b.getY() && b.getY() <= lavyDolny.getY() + dy)) {
                return false;
        }

        if (!(lavyDolny.getZ() <= b.getZ() && b.getZ() <= lavyDolny.getZ() + dz)) {
                return false;
        }

        return true;
}
```

Polymorfizmus je na to aby sme nemuseli používať instanceof.



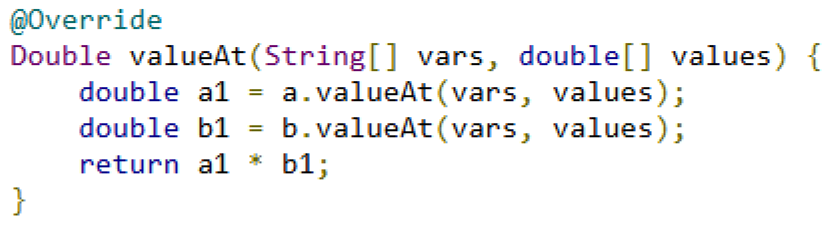
```java
@Override
Double valueAt(String[] vars, double[] values) {
    double product = 1;
    for (int j = 0; j < 2; j++) {
        if (operands[j] instanceof Konstanta) {
            product *= Double.parseDouble(operands[j].toString()
        } else if (operands[j] instanceof Premenna) {
            for (int i = 0; i < vars.length; i++) {
                if (vars[i] == operands[j].toString()) {
                    product *= values[i];
                    break;
                }
            }
        } else {
            product *= operands[j].valueAt(vars, values);
        }
    }

    return product;
}
```

👎



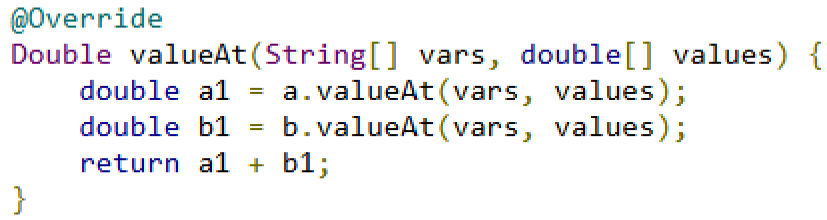
```java
@Override
Double valueAt(String[] vars, double[] values) {
    double sum = 0;
    for (int j = 0; j < 2; j++) {
        if (operands[j] instanceof Konstanta) {
            sum += Double.parseDouble(operands[j].toString());
        } else if (operands[j] instanceof Premenna) {
            for (int i = 0; i < vars.length; i++) {
                if (vars[i] == operands[j].toString()) {
                    sum += values[i];
                    break;
                }
            }
        } else {
            sum += operands[j].valueAt(vars, values);
        }
    }

    return sum;
}
```

👎



```java
@Override
Double valueAt(String[] vars, double[] values) {
    double a1 = a.valueAt(vars, values);
    double b1 = b.valueAt(vars, values);
    return a1 * b1;
}
```

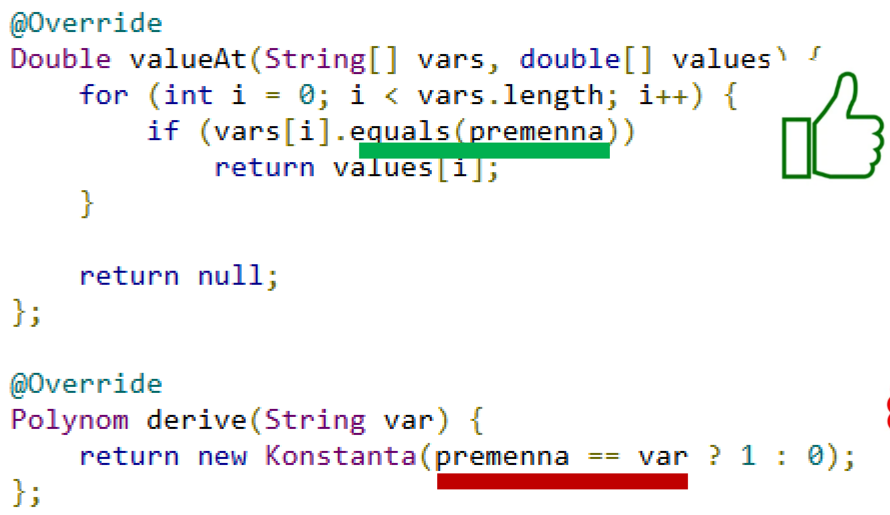
👍



```java
@Override
Double valueAt(String[] vars, double[] values) {
    double a1 = a.valueAt(vars, values);
    double b1 = b.valueAt(vars, values);
    return a1 + b1;
}
```

👍

String porovnávať vždy cez equals.



```
@Override
Double valueAt(String[] vars, double[] values) {
    for (int i = 0; i < vars.length; i++) {
        if (vars[i].equals(premenna))
            return values[i];
    }

    return null;
};

@Override
Polynom derive(String var) {
    return new Konstanta(premenna == var ? 1 : 0);
};
```

Porovnávanie double. Epsilon ideálne ako konštantu.

```java
/**
 * definujte test, ci bod b je v kvadri
 */
@Override
public boolean jeV(Bod3D b) {
    return (b.getX() > lavyDolny.getX() - 1e-5 && b.getX() < lavyDolny.getX() + dx + 1e-5)
            && (b.getY() > lavyDolny.getY() - 1e-5 && b.getY() < lavyDolny.getY() + dy + 1e-5)
            && (b.getZ() > lavyDolny.getZ() - 1e-5 && b.getZ() < lavyDolny.getZ() + dz + 1e-5);
}
```

```java
abstract class TriD {
        public static double EPSILON = 1e-9;
        public abstract double objem();
        public abstract double povrch();
        public abstract boolean jeV(Bod3D b);
        public abstract void posun(Bod3D b);
```

```java
@Override
public boolean jeV(Bod3D b) {
        double bx = b.getX();
        double by = b.getY();
        double bz = b.getZ();
        double lavyDolnyx = lavyDolny.getX();
        double lavyDolnyy = lavyDolny.getY();
        double lavyDolnyz = lavyDolny.getZ();

        return bx - lavyDolnyx >= -EPSILON && bx - lavyDolnyx - dx <= EPSILON &&
                        by - lavyDolnyy >= -EPSILON && by - lavyDolnyy - dy <= EPSILON &&
                        bz - lavyDolnyz >= -EPSILON && bz - lavyDolnyz - dz <= EPSILON;
}
```

Nevyťahovať do premenných veci čo už máme aj keď majú dlhý názov, len pre skrátenie riadku.
Radšej premennú pre každú dvojicu bX<x… a aj podmienka bude čitateľnejšia.

```java
@Override
public boolean jeV(Bod3D b) {
        double x = lavyDolny.getX();
        double y = lavyDolny.getY();
        double z = lavyDolny.getZ();

        double bX = b.getX();
        double bY = b.getY();
        double bZ = b.getZ();

        if (bX < x || bY < y || bX > x + dy || bY > y + dy || bZ < z || bZ > z + dz) {
                return false;
        }
        return true;
}
```

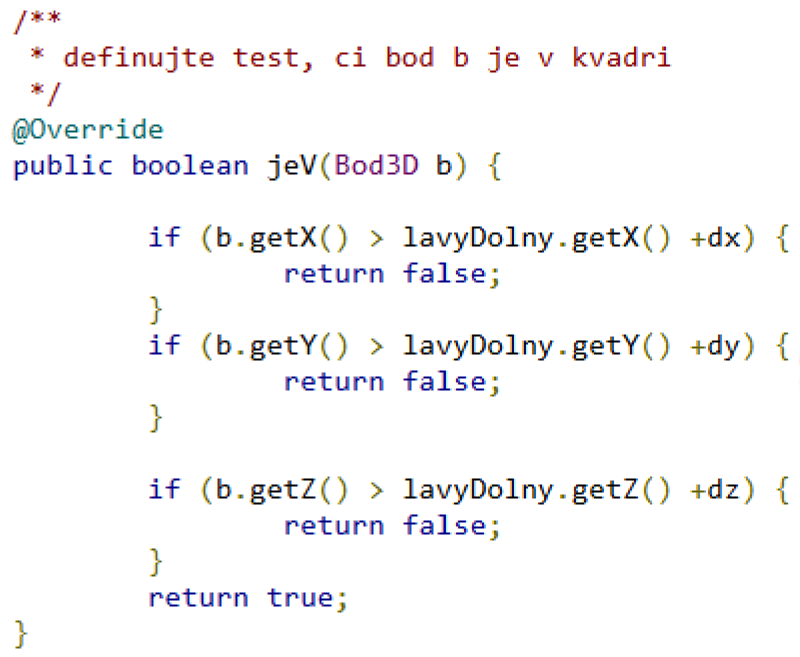
```java
@Override
public boolean jeV(Bod3D b) {
        double x = b.getX();
        double y = b.getY();
        double z = b.getZ();
        double x0 = lavyDolny.getX();
        double y0 = lavyDolny.getY();
        double z0 = lavyDolny.getZ();
        return x >= x0 && x <= x0 + dx && y >= y0 && y <= y0 + dy && z >= z0 && z <= z0 + dz;
}
```

Ctrl+c, Ctrl+v treba niekedy aj doplniť, nie len bez úvahy použiť.

```java
/**
 * definujte test, ci bod b je v obdlzniku, alebo na jeho stranach
 */
@Override
public boolean jeV(Bod2D b) {
        double x = b.getX();
        double y = b.getY();

        return x >= lavyDolny.getX() && x <= lavyDolny.getX() + dx &&
                       y >= lavyDolny.getY() && y <= lavyDolny.getY() + dy;
}
```

```java
/**
 * definujte test, ci bod b je v kvadri
 */
@Override
public boolean jeV(Bod3D b) {
        double x = b.getX();
        double z = b.getZ();

        return x >= lavyDolny.getX() && x <= lavyDolny.getX() + dx &&
                       z >= lavyDolny.getZ() && z <= lavyDolny.getZ() + dz;
}
```

Čo chýba?



```
/**
 * definujte test, ci bod b je v kvadri
 */
@Override
public boolean jeV(Bod3D b) {

        if (b.getX() > lavyDolny.getX() +dx) {
                return false;
        }
        if (b.getY() > lavyDolny.getY() +dy) {
                return false;
        }

        if (b.getZ() > lavyDolny.getZ() +dz) {
                return false;
        }
        return true;
}
```