```
/** Algorytm szukający miejsc zerowych za pomocą metody Laguerre'a
 *  Michał Ładanowski (c) 2011
 *  www.algorytm.org
 */

public class PolynomialRoots {
    // współczynniki danego wielomianu P
    private Complex[] polynomialCoeff;


    public PolynomialRoots(Complex[] polynomialCoeff){
        this.polynomialCoeff = polynomialCoeff;
    }

    // funkcja startująca algorytm
    public Complex[] findRoots(){
        Complex start = new Complex(0,0);
        Complex[] z = new Complex[polynomialCoeff.length -1];
        int i = 0;


        Complex[] polLess = null;
        z[0] = LaguerreMethod(start, polynomialCoeff);

        Complex[] tmp = polynomialCoeff;
        for(i = 1; i < polynomialCoeff.length -1; i++){
            polLess = newPol(z[i-1],tmp);
            z[i] = LaguerreMethod(start, polLess);
            z[i] = LaguerreMethod(z[i], polynomialCoeff);
            tmp = polLess;
        }


        for(i = 0; i < polynomialCoeff.length -1; i++){
            System.out.println(z[i]);
        }

        return z;
    }


    // Metoda Laguerre'a
    //  argumenty:
    //  start - punkt początkowy
    //  pol - tablica przechowująca współczynniki wielomianu,
       którego szukamy mz.
    //  zmienne:
    //  fun - P(z)*n
```

```
//  der - P'(z), secDer - P''(z)
//  denominator - mianownik
//  zwraca:
//  z - miejsce zerowe
private Complex LaguerreMethod(Complex start, Complex[] pol){
    Complex z = start;
    Complex fun,der,secDer, denominator;
    Complex tmp = new Complex(100,0);
    // iterujemy tak długo aż różnica między aktualnym a
       poprzednio
    // wyliczonym miejscem zerowym będzie nie mniejsca niż
    // zadana precyzja(w tym przypadku 1e-13)
    while((horner(z, pol).minus(horner(tmp, pol))).abs() >
       1e-13){
        tmp  = z;
        fun =horner(z, pol).times(pol.length-1);
        der = horner(z, derivative(pol));
        secDer = horner(z, derivative(derivative(pol)));
        denominator = ( (  der.times(der).times(((pol.length-1)
           - 1))
                .minus(fun.times(secDer)) ).times((pol.length-1)
                   - 1) ).sqrt();
        denominator = der.minus(denominator).abs() >
           der.plus(denominator).abs()
        ? der.minus(denominator) : der.plus(denominator);
        z = z.minus((fun.divides(denominator)));


    }

    return z;
}

// wartość wielomianu w z wyliczona za pomocą Hornera
private Complex horner(Complex z, Complex[] coefficients){
    Complex P = coefficients[coefficients.length-1];
    int k = coefficients.length-1;
    while(k >0){
        k--;
        P = P.times(z).plus(coefficients[k]);
    }
    return P;
}


// deflacja
public Complex[] newPol(Complex root, Complex[] oldPol){
    Complex[] newPol = new Complex[oldPol.length-1];
```

```
        newPol[oldPol.length -2] = oldPol[oldPol.length -1];
        for(int i = oldPol.length - 3; i >= 0; i--){
            newPol[i] = oldPol[i+1].plus(newPol[i+1].times(root));
        }


        return newPol;
    }

    // funkcja licząca pochodną
    public Complex[] derivative(Complex[] pol){
        if(pol.length-1 == 0) return new Complex[]{new
           Complex(0,0)};
        Complex[] der = new Complex[pol.length-1];

        for(int i = pol.length -2; i >= 0; i-- ){
            der[i] = pol[i+1].times(i+1);
        }

        return der;
    }


}
```