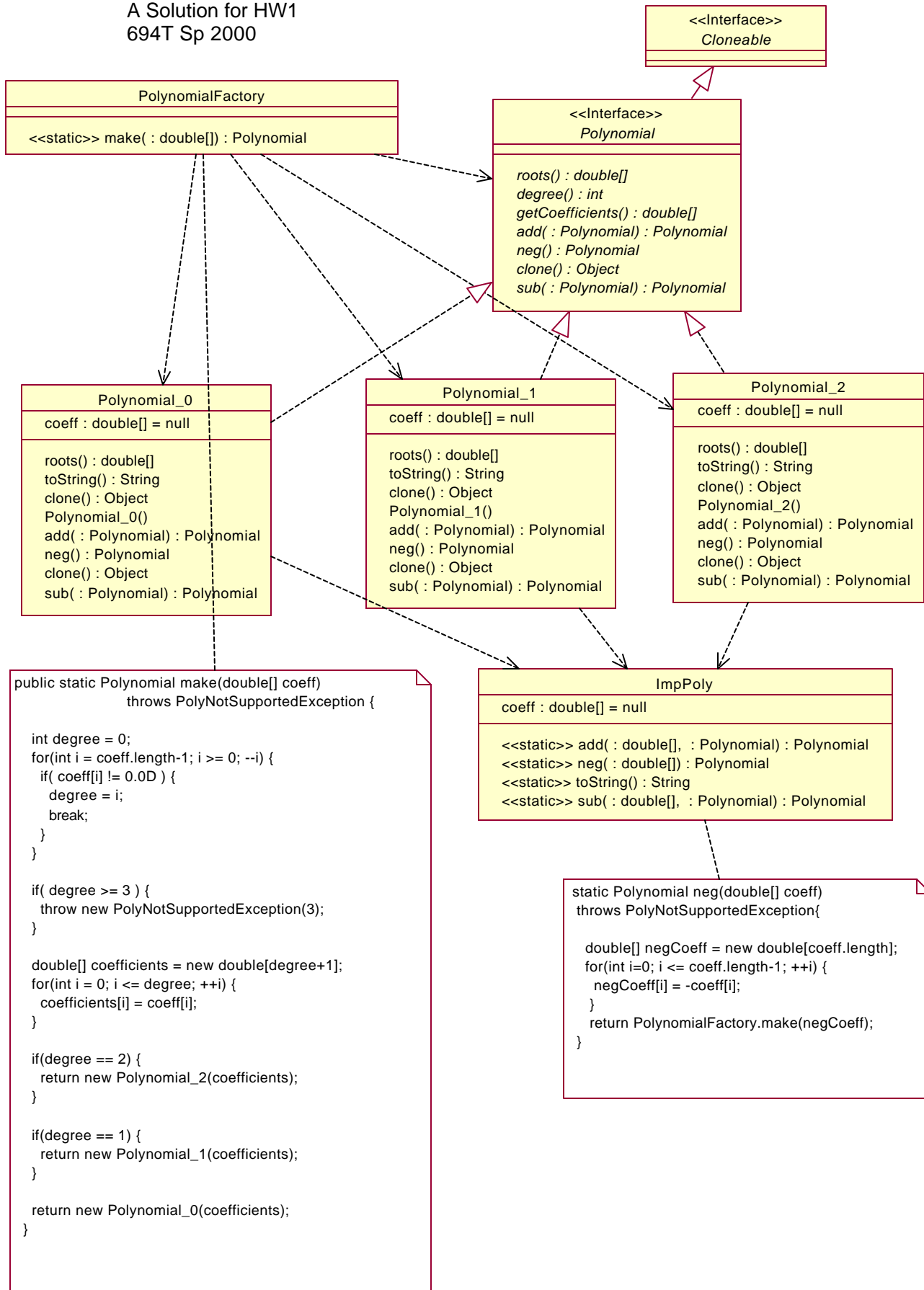
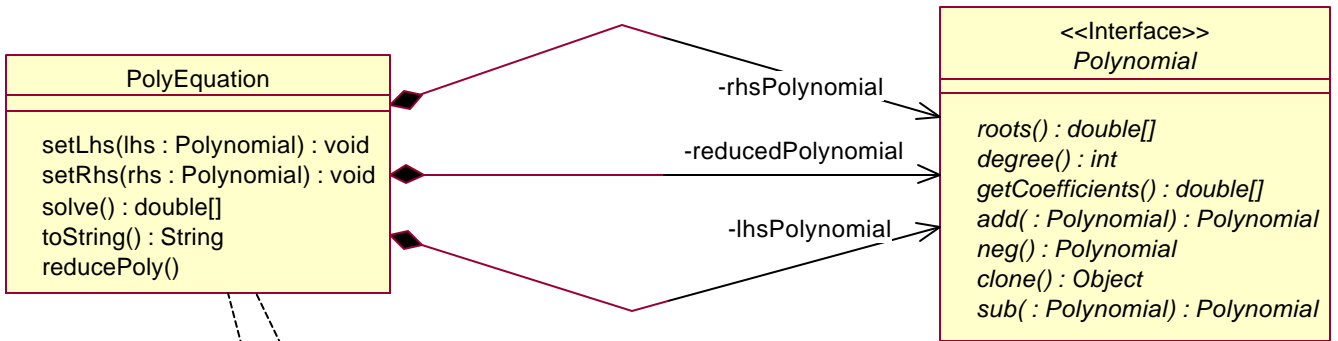


A Solution for HW1
694T Sp 2000





```
private void reducePoly() throws PolyNotSupportedException {
    reducedPolynomial = lhsPolynomial.sub(rhsPolynomial);
}
```

```
public double[] solve() {
    return reducedPolynomial.realRoots();
}
```

A Solution for HW1 694T Spring 2000

```
package polyDemo;
import polynomials.*;

public class Client {

    public static void main(String[] args) throws
        PolyNotSupportedException{
        double[] coeff = new double[5];

        System.out.println("----- clone test -----");
        coeff[0] = 0.1D;
        coeff[1] = -2.0D;
        coeff[2] = 8.0D;
        coeff[3] = 0.0D;
        coeff[4] = 0.0D;

        Polynomial poly1 = PolynomialFactory.make(coeff);
        Polynomial poly2 = (Polynomial) poly1.clone();

        System.out.println("poly1: "+poly1);
        System.out.println("poly2, clone of poly1: "+poly2);

        System.out.println("----- subtraction test -----");

        coeff[0] = 0.2D;
        coeff[1] = -2.1D;
        coeff[2] = 8.0D;
        coeff[3] = 0.0D;

        Polynomial poly3 = PolynomialFactory.make(coeff);
        Polynomial poly4 = poly1.sub(poly3);

        System.out.println("poly1: "+poly1);
        System.out.println("poly3: "+poly3);
        System.out.println("poly1-poly3: "+poly4);

        System.out.println("----- equation creation test1 -----");

        System.out.println("poly1 :"+poly1);
        System.out.println("poly3: "+poly3);
        Equation eq = new Equation(poly1, poly3);
        System.out.println("Equation, poly1 = poly3: "+eq);
        System.out.println("Reduced poly of equation poly1-poly3:
"+eq.getReducedPoly());

        System.out.println("----- equation creation test2 -----");

        System.out.println("poly1: "+poly1);
        eq = new Equation(poly1);
        System.out.println("Equation, poly1 = 0: "+eq);
```

```

    System.out.println("Reduced poly of equation poly1 = 0:
"+eq.getReducedPoly());

    System.out.println("---- roots test1 -----");

    coeff[0] = 1.0;
    coeff[1] = -2.0D;
    coeff[2] = 1.0D;
    poly1 = PolynomialFactory.make(coeff);
    System.out.println("poly1: "+poly1);
    double[] roots = poly1.realRoots();
    System.out.print("Roots of poly1 are:");
    if(roots != null)
    {
        for(int i = 0; i < roots.length; ++i) {
            System.out.print(roots[i]);
        }
    } else {
        System.out.println("poly1 has no roots");
    }

    System.out.println("---- roots test2 -----");

    coeff[0] = 1.0;
    coeff[1] = 2.0D;
    coeff[2] = 4.0D;
    poly1 = PolynomialFactory.make(coeff);
    System.out.println("poly1: "+poly1);
    roots = poly1.realRoots();
    System.out.print("Roots of poly1 are:");
    if(roots != null)
    {
        for(int i = 0; i < roots.length; ++i) {
            System.out.print(roots[i]);
        }
    } else {
        System.out.println("poly1 has no roots");
    }
}
}

```

```

package polynomials;

```

```

public final class Equation {

```

```

    private Polynomial lhsPolynomial = null;
    private Polynomial rhsPolynomial = null;
    private Polynomial reducedPolynomial = null;

```

```

    public Equation(Polynomial lhs, Polynomial rhs) throws
        PolyNotSupportedException{
        lhsPolynomial = (Polynomial) lhs.clone();
        rhsPolynomial = (Polynomial) rhs.clone();
        reducePoly();
    }

```

```

    }

    public Equation(Polynomial lhs) throws PolyNotSupportedException{
        lhsPolynomial = (Polynomial) lhs.clone();
        double[] rhsCoeff = new double[1];
        rhsCoeff[0] = 0.0D;
        rhsPolynomial = PolynomialFactory.make(rhsCoeff);
        reducedPolynomial = (Polynomial) lhsPolynomial.clone();
    }

    public Equation() throws PolyNotSupportedException{
        double[] coeff = new double[1];
        coeff[0] = 0.0D;
        lhsPolynomial = PolynomialFactory.make(coeff);
        rhsPolynomial = PolynomialFactory.make(coeff);
        reducedPolynomial = PolynomialFactory.make(coeff);
    }

    public void setLhs(Polynomial lhs) throws PolyNotSupportedException{
        lhsPolynomial = (Polynomial) lhs.clone();
        reducePoly();
    }

    public void setRhs(Polynomial rhs) throws PolyNotSupportedException{
        rhsPolynomial = (Polynomial) rhs.clone();
        reducePoly();
    }

    public Polynomial getReducedPoly() {
        return (Polynomial) reducedPolynomial.clone();
    }

    public String toString() {
        String outString = "["+lhsPolynomial+" = "+rhsPolynomial+"]";
        return outString;
    }

    public double[] solve() {
        return reducedPolynomial.realRoots();
    }

    private void reducePoly() throws PolyNotSupportedException{
        reducedPolynomial = lhsPolynomial.sub(rhsPolynomial);
    }
}

```

package polynomials;

```

public interface Polynomial extends Cloneable{
    public int degree();
    public Object clone();
    public Polynomial add(Polynomial polyToAdd) throws
        PolyNotSupportedException;
    public Polynomial sub(Polynomial polyToAdd) throws
        PolyNotSupportedException;
}

```

```
public Polynomial neg() throws PolyNotSupportedException;
public double[] getCoeff();
public double[] realRoots();
}
```

package polynomials;

```
final class Polynomial_0 implements Polynomial{

    private double[] coeff = new double[1];

    private Polynomial_0() {
        super();
    }

    Polynomial_0(double[] coeff) {
        this.coeff = coeff;
    }

    public int degree() {
        return coeff.length-1;
    }

    public double[] getCoeff() {
        return (double[]) coeff.clone();
    }

    public Polynomial add(Polynomial polyToAdd)
        throws PolyNotSupportedException{
        return ImplPoly.add(coeff, polyToAdd);
    }

    public Polynomial sub(Polynomial polyToSub)
        throws PolyNotSupportedException{
        return add( polyToSub.neg() );
    }

    public Polynomial neg() throws PolyNotSupportedException {
        return ImplPoly.neg(coeff);
    }

    public double[] realRoots() {
        return null;
    }

    public String toString() {
        return ImplPoly.toString(coeff);
    }

    public Object clone() {
        try {
            return super.clone();
        }
        catch (CloneNotSupportedException e) {
```

```

        // we should never reach here since Polynomial extends
        // Cloneable
        return null;
    }
}

```

package polynomials;

```

final class Polynomial_1 implements Polynomial{

    private double[] coeff = null;

    private Polynomial_1() {
        super();
    }

    Polynomial_1(double[] coeff) {
        this.coeff = (double[])coeff.clone();
    }

    public int degree() {
        return coeff.length-1;
    }

    public double[] getCoeff() {
        return (double[]) coeff.clone();
    }

    public Polynomial add(Polynomial polyToAdd)
        throws PolyNotSupportedException{
        return ImplPoly.add(coeff, polyToAdd);
    }

    public Polynomial sub(Polynomial polyToSub)
        throws PolyNotSupportedException{
        return add( polyToSub.neg() );
    }

    public Polynomial neg() throws PolyNotSupportedException {
        return ImplPoly.neg(coeff);
    }

    public double[] realRoots() {
        double[] root = new double[1];
        root[0]= -coeff[0]/coeff[1];
        return root;
    }

    public String toString() {
        return ImplPoly.toString(coeff);
    }

    public Object clone() {
        try {

```

```

        return super.clone();
    }
    catch (CloneNotSupportedException e) {
        // we should never reach here since Polynomial extends
        // Cloneable
        return null;
    }
}
}
}

```

package polynomials;

final class Polynomial_2 implements Polynomial{

```

    private double[] coeff = null;

    private Polynomial_2() {
        super();
    }

    Polynomial_2(double[] coeff) {
        this.coeff = coeff;
    }

    public int degree() {
        return coeff.length-1;
    }

    public double[] getCoeff() {
        return (double[]) coeff.clone();
    }

    public Polynomial add(Polynomial polyToAdd)
        throws PolyNotSupportedException{
        return ImplPoly.add(coeff, polyToAdd);
    }

    public Polynomial sub(Polynomial polyToSub)
        throws PolyNotSupportedException{
        return add( polyToSub.neg() );
    }

    public Polynomial neg() throws PolyNotSupportedException {
        return ImplPoly.neg(coeff);
    }

    public double[] realRoots() {
        double det = coeff[1]*coeff[1]-4.0D*coeff[2]*coeff[0];

        if(det == 0.0D) {
            double[] root = new double[1];
            root[0] = -0.5D*coeff[1]/coeff[2];
            return root;
        }
        if(det > 0.0D) {
            double[] root = new double[2];

```

```

        double sqrtDet = Math.sqrt(det);
        root[0] = -0.5D*(coeff[1]+sqrtDet)/coeff[2];
        root[1] = 0.5D*(-coeff[1]+sqrtDet)/coeff[2];
        return root;
    }

    return null;
}

public String toString() {
    return ImplPoly.toString(coeff);
}

public Object clone() {
    try {
        return super.clone();
    }
    catch (CloneNotSupportedException e) {
        // we should never reach here since Polynomial extends
        // Cloneable
        return null;
    }
}
}
}

```

package polynomials;

final class ImplPoly {

```

    static String toString(double[] coeff){

        String outString = "Degree: "+(coeff.length-1)+", Parameters: ";

        for (int i = 0; i <= coeff.length-1; ++i) {
            outString = outString+coeff[i];
            if(i != coeff.length - 1 ) {
                outString = outString +", ";
            }
        }

        return outString;
    }

    static Polynomial add(double[] coeff, Polynomial polyToAdd)
    throws PolyNotSupportedException{

        double[] polyToAddCoeff = polyToAdd.getCoeff();

        double[] large = null;
        double[] small = null;
        if(coeff.length >= polyToAddCoeff.length) {
            large = coeff;
            small = polyToAddCoeff;
        }
        else {
            large = polyToAddCoeff;

```

```

    small = coeff;
}

double[] addCoeff = (double[]) large.clone();
for(int i=0; i <= small.length-1; ++i) {
    addCoeff[i] = addCoeff[i]+small[i];
}
return PolynomialFactory.make(addCoeff);
}

static Polynomial neg(double[] coeff)
throws PolyNotSupportedException{

    double[] negCoeff = new double[coeff.length];
    for(int i=0; i <= coeff.length-1; ++i) {
        negCoeff[i] = -coeff[i];
    }
    return PolynomialFactory.make(negCoeff);
}
}

```

```

package polynomials;

```

```

public final class PolynomialFactory {

```

```

    public static Polynomial make(double[] coeff) throws
    PolyNotSupportedException {

```

```

        int degree = 0;
        for(int i = coeff.length-1; i >= 0; --i) {
            if( coeff[i] != 0.0D ) {
                degree = i;
                break;
            }
        }
    }

```

```

    if( degree >= 3 ) {
        throw new PolyNotSupportedException(3);
    }

```

```

    double[] coefficients = new double[degree+1];
    for(int i = 0; i <= degree; ++i) {
        coefficients[i] = coeff[i];
    }

```

```

    if(degree == 2) {
        return new Polynomial_2(coefficients);
    }

```

```

    if(degree == 1) {
        return new Polynomial_1(coefficients);
    }

```

```

    return new Polynomial_0(coefficients);
}

```

```
}
```

```
package polynomials;
```

```
public class PolyNotSupportedException extends Exception{
```

```
    PolyNotSupportedException(int degree) {  
        super(" [Ploynomials with degree >= "+degree+" not supported]");  
    }  
}
```

```
}
```

```
---- clone test -----
```

```
poly1: Degree: 2, Parameters: 0.1, -2.0, 8.0
```

```
poly2, clone of poly1: Degree: 2, Parameters: 0.1, -2.0, 8.0
```

```
---- subtraction test -----
```

```
poly1: Degree: 2, Parameters: 0.1, -2.0, 8.0
```

```
poly3: Degree: 2, Parameters: 0.2, -2.1, 8.0
```

```
poly1-poly3: Degree: 1, Parameters: -0.1, 0.100000000000000009
```

```
---- equation creation test1 -----
```

```
poly1 :Degree: 2, Parameters: 0.1, -2.0, 8.0
```

```
poly3: Degree: 2, Parameters: 0.2, -2.1, 8.0
```

```
Equation, poly1 = poly3: [Degree: 2, Parameters: 0.1, -2.0, 8.0 =
```

```
Degree: 2, Parameters: 0.2, -2.1, 8.0]
```

```
Reduced poly of equation poly1-poly3: Degree: 1, Parameters: -0.1,  
0.100000000000000009
```

```
---- equation creation test2 -----
```

```
poly1: Degree: 2, Parameters: 0.1, -2.0, 8.0
```

```
Equation, poly1 = 0: [Degree: 2, Parameters: 0.1, -2.0, 8.0 = Degree:  
0, Parameters: 0.0]
```

```
Reduced poly of equation poly1 = 0: Degree: 2, Parameters: 0.1, -2.0,  
8.0
```

```
---- roots test1 -----
```

```
poly1: Degree: 2, Parameters: 1.0, -2.0, 1.0
```

```
Roots of poly1 are:1.0
```

```
---- roots test2 -----
```

```
poly1: Degree: 2, Parameters: 1.0, 2.0, 4.0
```

```
Roots of poly1 are:poly1 has no roots
```