```java
package s12pruefung2.v1;
import java.util.*;
import s12pruefung2.*;

public class Polygon implements Cloneable {
    private Punkt[] punkte;
    public Polygon(Punkt... punkte) {
        if (punkte.length < 2) throw new IllegalArgumentException();
        this.punkte = punkte;
    }
    public int anzahlPunkte() {
        return punkte.length;
    }
    public Punkt anfangsPunkt() {
        return punkte[0];
    }
    public Punkt endPunkt() {
        return punkte[anzahlPunkte() - 1];
    }
    public Punkt[] punkte() {
        return punkte.clone();
    }
    public Polygon geschlossen() {
        if (istGeschlossen()) return this;
        Punkt[] neu = Arrays.copyOf(punkte, anzahlPunkte() + 1);
        neu[anzahlPunkte()] = punkte[0];
        return new Polygon(neu);
    }
    public boolean istGeschlossen() {
        return anfangsPunkt().equals(endPunkt());
    }
    public boolean equals(Object o) {
        if (!(o instanceof Polygon)) return false;
        Polygon p = (Polygon) o;
        if (p.anzahlPunkte() != anzahlPunkte()) return false;
        for (int i = 0; i < anzahlPunkte(); i++)
            if (p.punkte[i] != punkte[i]) return false;
        return true;
    }
    public int hashCode() {
        return anzahlPunkte();
    }
}
```

```java
package s12pruefung2.v2;
import java.util.*;
import s12pruefung2.Punkt;

public final class Polygon {
    private static final Punkt[] LEER = new Punkt[0];
    private final List<Punkt> mPunkte;
    public Polygon(Punkt erster, Punkt zweiter, Punkt... weitere) {
        mPunkte = new ArrayList<Punkt>();
        mPunkte.add(erster);
        mPunkte.add(zweiter);
        for (Punkt p : weitere) mPunkte.add(p);
    }
    public Polygon(List<Punkt> punkte) {
        if (punkte.size() < 2) throw new IllegalArgumentException();
        mPunkte = punkte;
    }
    public int anzahlPunkte() {
        return mPunkte.size();
    }
    public Punkt anfangsPunkt() {
        return mPunkte.get(0);
    }
    public Punkt endPunkt() {
        return mPunkte.get(anzahlPunkte() - 1);
    }
    public Polygon geschlossen() {
        if (istGeschlossen()) return this;
        List<Punkt> neu = new ArrayList<Punkt>(mPunkte);
        neu.add(anfangsPunkt());
        return new Polygon(neu);
    }
    public boolean istGeschlossen() {
        return anfangsPunkt().equals(endPunkt());
    }
    public Punkt[] punkte() {
        return mPunkte.toArray(LEER);
    }
    public boolean equals(Polygon p) {
        if (p == this) return true;
        if (p == null) return false;
        return p.mPunkte.equals(mPunkte);
    }
}
```

```java
package s12pruefung2.v3;
import static java.lang.Math.*;
import java.util.*;
import s12pruefung2.*;

public final class Polygon extends Punkt {
    protected List<Punkt> mPunkte = new ArrayList<Punkt>();
    public Polygon(Punkt erster, Punkt... weitere) {
        super(erster.x(), erster.y());
        mPunkte = new ArrayList<Punkt>();
        mPunkte.add(erster);
        for (Punkt p : weitere) mPunkte.add(p);
    }
    private Polygon(List<Punkt> punkte) {
        super(punkte.get(0).x(), punkte.get(0).y());
        mPunkte = punkte;
    }
    public int anzahlPunkte() {
        return mPunkte.size();
    }
    public Punkt anfangsPunkt() {
        return this;
    }
    public Punkt endPunkt() {
        return mPunkte.get(anzahlPunkte() - 1);
    }
    public Punkt[] punkte() {
        return (Punkt[]) mPunkte.toArray(new Punkt[0]);
    }
    public Polygon geschlossen() {
        if (istGeschlossen()) return this;
        List<Punkt> neu = new ArrayList<Punkt>(mPunkte);
        neu.add(anfangsPunkt());
        return new Polygon(neu);
    }
    public boolean istGeschlossen() {
        return anfangsPunkt().equals(endPunkt());
    }
    public double abstandZu(Punkt p) {
        double erg = Double.MAX_VALUE;
        for (Punkt q : mPunkte) erg = min(erg, q.abstandZu(p));
        return erg;
    }
}
```

```java
package s12pruefung2.v4;
import java.util.*;
import s12pruefung2.*;

public final class Polygon extends ArrayList<Punkt> {
    private static final long serialVersionUID = 1L;
    public Polygon(Punkt... punkte) {
        for (Punkt p : punkte) add(p);
    }
    private Polygon(List<Punkt> punkte) {
        for (Punkt p : punkte) add(p);
    }
    public Punkt anfangsPunkt() {
        return get(0);
    }
    public Punkt endPunkt() {
        return get(size() - 1);
    }
    public Polygon geschlossen() {
        if (istGeschlossen()) return this;
        List<Punkt> neu = new ArrayList<Punkt>(this);
        neu.add(anfangsPunkt());
        return new Polygon(neu);
    }
    public boolean istGeschlossen() {
        return anfangsPunkt().equals(endPunkt());
    }
    public Punkt[] punkte() {
        return (Punkt[]) toArray(new Punkt[0]);
    }
}
```