```
package anhang;
public enum Spalte { A, B, C, D, E, F, G }
```

```
package anhang;
public enum Farbe {
    ROT { public Farbe andere() { return GELB; }
          public char kuerzel() { return 'R'; } },
    GELB { public Farbe andere() { return ROT; }
           public char kuerzel() { return 'G'; } },
    LEER { public Farbe andere() { throw new InternalError(); }
           public char kuerzel() { return '-'; } };
    public abstract Farbe andere();
    public abstract char kuerzel();
}
```

```
package anhang;
import java.util.*;

public class StellungsEntwicklung extends Beobachtet {
    private final List<Spalte> mSpalten = new ArrayList<Spalte>();
    private final Farbe mBeginn;
    private Stellung mStellung;
    public StellungsEntwicklung(Farbe beginn, Spalte... spalten) {
        mBeginn = beginn;
        for (Spalte s : spalten) mSpalten.add(s);
    }
    public String toString() {
        Farbe amZuge = mBeginn;
        if (halbzuege() % 2 != 0) amZuge = mBeginn.andere();
        return "" + mBeginn + "->" +  mSpalten + "->" + amZuge;
    }
    public int halbzuege() { return mSpalten.size(); }
    public Spalte letzterZug() {
        if (halbzuege() == 0) return null;
        return mSpalten.get(mSpalten.size() - 1);
    }
    public StellungsEntwicklung einwerfenBei(Spalte... spalten) {
        if (spalten.length > 0) mStellung = null;
        for (Spalte s : spalten) mSpalten.add(s);
        nachrichtAnAlle();
        return this;
    }
    public Stellung stellung() {
        if (mStellung == null) mStellung =
                new Stellung(mBeginn, mSpalten.toArray(new Spalte[0]));
        return mStellung;
    }
}
```

```
package anhang;
public enum Richtung {
    O(1,0),NO(1,1),N(0,1),NW(-1,1),W(-1,0),SW(-1,-1),S(0,-1),SO(1,-1);
    private final int mDx, mDy;
    private Richtung(int dx, int dy) { mDx = dx; mDy = dy; }
    public int inX() { return mDx; }    public int inY() { return mDy; }
}
```

```
package anhang.beobachtung;
import anhang.*;
import static anhang.Stellung.*;

public final class DarStellung implements Beobachter {
    private static int mBeobachter = 0;
    private final int mNummer = ++mBeobachter;
    private Stellung mLetzte;
    public DarStellung() { }
    public void aktualisiere(Beobachtet b) {
        StellungsEntwicklung s = (StellungsEntwicklung) b;
        if (s.stellung().equals(mLetzte)) return;
        Farbe[][] matrix = s.stellung().matrix();
        for (int i = ANZAHL_REIHEN - 1; i >= 0; i--) {
            System.out.println();
            for (int j = 0; j < ANZAHL_SPALTEN; j++)
                System.out.print(" " + matrix[i][j].kuerzel());
            if (i == 3) System.out.print("  Beobachter: #" + mNummer);
            if (i == 1) System.out.print("  Halbzuege: " + s.halbzuege());
        }
        System.out.println("  am Zuge: " + s.stellung().amZuge());
    }
    public static void darstellen(Beobachtet b) {
       b.anmelden(new DarStellung());
    }
}
```

```
package anhang;
import java.util.*;

public abstract class Beobachtet {
    private final Set<Beobachter> mAnmeldungen = new HashSet<>();
    public final void anmelden(Beobachter sb) {
        mAnmeldungen.add(sb);
        sb.aktualisiere(this);
    }
    public final void nachrichtAnAlle() {
        for (Beobachter sb : mAnmeldungen) sb.aktualisiere(this);
    }
}
```

```
package anhang;
public interface Beobachter {
    void aktualisiere(Beobachtet b);
}
```

```
package anhang;
import java.util.*;
import static java.util.Arrays.*;
import static anhang.Farbe.*;

public final class Stellung implements java.io.Serializable {
    public final static int ANZAHL_REIHEN = 6;
    public static final int ANZAHL_SPALTEN = Spalte.values().length;
    private Farbe[][] mMatrix = new Farbe[ANZAHL_REIHEN][ANZAHL_SPALTEN];
    private final Farbe mAmZuge;
    public Farbe amZuge() { return mAmZuge; }
    public Farbe[][] matrix() { return mMatrix; }
    public Stellung(Farbe beginn, Spalte... spalten) {
        for (int i = 0; i < ANZAHL_REIHEN; i++) fill(mMatrix[i], LEER);
        Farbe f = beginn;
        for (Spalte s : spalten) {
            mMatrix[ANZAHL_REIHEN - nochFreiBei(s)][s.ordinal()] = f;
            f = f.andere();
        }
        mAmZuge = f;
    }
    public Stellung(Stellung basis, Spalte s) {
        mAmZuge = basis.mAmZuge.andere();
        mMatrix = basis.mMatrix.clone();
        int i = ANZAHL_REIHEN - nochFreiBei(s);
        mMatrix[i] = basis.mMatrix[i].clone();
        mMatrix[i][s.ordinal()] = basis.mAmZuge;
    }
    public final int nochFreiBei(Spalte s) {
        for (int i = 0; i < ANZAHL_REIHEN; i++)
            if (mMatrix[i][s.ordinal()] == LEER) return ANZAHL_REIHEN - i;
        return 0;
    }
    public Spalte zufaelligeSpalte() {
        List<Spalte> liste = asList(Spalte.values());
        Collections.shuffle(liste);
        for (Spalte s : liste) if (nochFreiBei(s) != 0) return s;
        throw new IllegalStateException();
    }
    @Override public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null) return false;
        if (getClass() != o.getClass()) return false;
        Stellung s = (Stellung) o;
        for (int i = 0; i < ANZAHL_REIHEN; i++)
            if (!Arrays.equals(mMatrix[i], s.mMatrix[i])) return false;
        return true;
    }
    private boolean istLeer(int reihe) {
        for (int j = 0; j < ANZAHL_SPALTEN; j++)
            if (mMatrix[reihe][j] != LEER) return false;
        return true;
    }
    public String toString() {
        String erg = "|";
        for (int i = 0; i < ANZAHL_REIHEN; i++) {
            if (istLeer(i)) break;
            for (int j = 0; j < ANZAHL_SPALTEN; j++)
                erg += mMatrix[i][j].kuerzel();
            erg += "|";
        }
        return erg + "" + mAmZuge;
    }
    public int anzahlGleiche(Spalte s, int reihe, Richtung ri) {
        if (reihe <= 0 || reihe > ANZAHL_REIHEN)
                                throw new IllegalArgumentException();
        int i = reihe - 1; int j = s.ordinal();
        Farbe basis = mMatrix[i][j];
        int erg = 1;
        for (int n = 0; n <= ANZAHL_SPALTEN; n++) {
            i += ri.inY(); j += ri.inX();
            if (!istLegal(i, j) || mMatrix[i][j] != basis) return erg;
            erg++;
        }
        throw new InternalError();
    }
    private static boolean istLegal(int i, int j) {
        return i>=0 && i<ANZAHL_REIHEN && j>=0 && j<ANZAHL_SPALTEN;
    }
    public int anzahlGleiche(Spalte s, int reihe) {
        int erg = 0;
        for (Richtung ri : Richtung.values()) {
            int anzahl = anzahlGleiche(s, reihe, ri);
            if (anzahl > erg) erg = anzahl;
        }
        return erg;
    }
}
```