Igra `TicTacToe` (logika)

Razred `Object`

Vsak razred je podrazed razreda `Object`

`Objekt` razred ima metode

```
String toString()

boolean equals(Object o)

int hashCode()
```

Če razred ne definira teh metod, podeduje privzete metode iz razreda `Object`.

```
public class Koordinati {
    private int x;
    private int y;

    public Koordinati(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() { return x; }
    public int getY() { return y; }
}
```

Primer:

```
Koordinati h = new Koordinati(2,3);
Koordinati k = new Koordinati(2,3);

h.equals(k) ==> false  // objekta nista enaka
h.toString() ==> "Koordinati@7852e922"   // referenca objekta
h.hashCode() ==> 1311053135
k.hashCode() ==> 2018699554  // x in y imata drugi kodi
```

Dodamo na razred `Koordinati`

```
@Override
public String toString() {
    return "Koordinati [x=" + x + ", y=" + y + "]";
}

@Override
public boolean equals(Object o) { // tip parametra o mora biti Object
    if (this == o) return true;  // hiter in pogost primer
    if (o == null || this.getClass() != o.getClass()) return false;
    Koordinati k = (Koordinati) o; // vemo, da je Koordinati tip objekta o
    return this.x == k.x && this.y == k.y;
}

@Override
public int hashCode () {
    return Objects.hash(this.x, this.y);
    }
```

```
Koordinati h = new Koordinati(2,3);
Koordinati k = new Koordinati(2,3);

(h == k) ==> false  // objekta nista ista
h.equals(k) ==> true  // ampak sta enaka
h.toString() ==> "Koordinati [x=3, y=2]"
h.hashCode() ==> 1026
k.hashCode() ==> 1026
```

Če preglasimo metodo `equals`, je priporočljivo, da prelgasimo tudi metodo `hashCode`.

Te dve metodi morata izpolniti pogoj:

> če sta objekta `x` in `y` enaki (tj. če `x.equals(y) == true`), potem sledi `x.hashCode() == y.hashCode()`.

(V resnici, je metoda `hashCode` pomembna samo, če bomo uporabjali objekte razrada kot ključe za 'hash table'.)

Nekaj razredov ima že dobro metodo `equals`.

```
String x = "abc";
String y = "abc" + "";
String z = x + "";

(x=="abc") ==> true
(x==y) ==> true
(y==z) ==> false   // pazite!
y.equals(z) ==> true
```

Pomembno je, da uporabljamo `equals` namesto `==`, kadar primerjamo nize.

Tipi enum

Tip za igralca v igri tic-tac-toe

```
public enum Igralec {
    O, X;

    public Igralec nasprotnik() {
        return (this == X ? O : X);
    }

    public Polje getPolje() {
        return (this == X ? Polje.X : Polje.O);
    }
}
```

`Igralec` je referenčni tip s (statičnima) vrednostma `O` in `X`, ter metodi `nasprotnik` in `getPolje`.

Tip za vrednosti polj na plošči.

```
public enum Polje {
    O, X, PRAZNO
}
```

Vsak igralec je računalnik ali človek.

```
enum VrstaIgralca { R, C; }

Map<Igralec,VrstaIgralca> vrstaIgralca;
```

Spremljivka `vrstaIgralca` je map (slovar), ki da vsakemu igralcu (O ali X) njegovo vrsto (računalnik ali človek).

Na splošno, `Map<K,V>` je vmesnik za razrede, ki implementirajo slovare (maps), ki imajo kjluče tipa `K` in vrednosti tipa `V`.

- `HashMap<K,V>` implementira `Map<K,V>` za razrede `K`, ki imajo dobre implementacije metod `equals` in `hashCode`. V tem primeru, je slovar implementiran kot 'hash table'.
- `EnumMap<K,V>` implementira `Map<K,V>` za razrede `K`, v katerih je `K` enum tip.

```
private static BufferedReader r =
    new BufferedReader(new InputStreamReader(System.in));

public static void igramo () throws IOException {
    while (true) {
        System.out.println("Nova igra. Prosim, da izberete:");
        System.out.println(" 1 - O clovek, X računalnik");
        System.out.println(" 2 - O računalnik, X človek");
        System.out.println(" 3 - O človek, X človek");
        System.out.println(" 4 - izhod");
        String s = r.readLine();
        if (s.equals("1")) {
            vrstaIgralca = new EnumMap<Igralec,VrstaIgralca>(Igralec.class);
            vrstaIgralca.put(Igralec.O, VrstaIgralca.C);
            vrstaIgralca.put(Igralec.X, VrstaIgralca.R);
        } else if (s.equals("2")) { ... podobno ...}
        } else if (s.equals("3")) { ... podobno ...}
        } else if (s.equals("4")) {
            System.out.println("Nasvidenje!");
            break;  // izstopimo iz zanke
        } else {
            System.out.println("Vnos ni veljaven");
            continue; // gremo na naslednje okrog zanke
        }
        // Ce je s == "1", "2" ali "3"
        // koda na naslednji strani gre tukaj
    }
}
```

```
Igra igra = new Igra ();
igranje : while (true) {
    switch (igra.stanje()) {
    case ZMAGA_O:
        System.out.println("Zmagal je igralec O");
        break igranje;
    case ZMAGA_X:
        System.out.println("Zmagal je igralec X");
        break igranje;
    case NEODLOCENO:
        System.out.println("Igra je neodločena");
        break igranje;
    case V_TEKU:
        Igralec igralec = igra.naPotezi();
        VrstaIgralca vrstaNaPotezi = vrstaIgralca.get(igralec);
        Koordinati poteza = null;
        switch (vrstaNaPotezi) {
        case C:
            poteza = clovekovaPoteza(igra);
            break;
        case R:
            poteza = racunalnikovaPoteza(igra);
            break;
        }
        System.out.println("Igralec " + igralec + " je igral " + poteza);
    }
}
```

Tip za stanje igre

```
enum Stanje {
    V_TEKU, ZMAGA_X, ZMAGA_O, NEODLOCENO;
}
```

Lahko uporabljamo izjavo `switch`, ko imamo spremenljivke tipov `enum` ali tipa `int`, `short`, `char`, `byte`, `String`.

Če ne pišemo `break` v izjavi `case`, Java nadaljuje z nadlednjim izjavo.

Na prejšnji strani uporabljamo tudi `break igranje`, kjer je `igranje` nalepka.

Metoda racunalnikovaPoteza

```
private static Random random = new Random ();

public static Koordinati racunalnikovaPoteza(Igra igra) {
    List<Koordinati> moznePoteze = igra.poteze();
    int randomIndex = random.nextInt(moznePoteze.size());
    Koordinati poteza = moznePoteze.get(randomIndex);
    igra.odigraj(poteza);
    return poteza;
}
```

Metoda clovekovaPoteza

```
public static Koordinati clovekovaPoteza(Igra igra) throws IOException {
    while (true) {
        Igralec igralec = igra.naPotezi();
        System.out.println(igralec + " vnesite potezo \"x y\"");
        String s = r.readLine();
        int i = s.indexOf (' '); // kje je presledek
        if (i == -1 || i  == s.length()) {
            System.out.println("Napačen format"); continue;
        }
        String xString = s.substring(0,i);
        String yString = s.substring(i+1);
        int x, y;
        try {
            x = Integer.parseInt(xString);
            y = Integer.parseInt(yString);
        } catch (NumberFormatException e) {
            System.out.println("Napačen format"); continue;
        }
        if (x < 0 || x >= Igra.N || y < 0 || y >= Igra.N){
            System.out.println("Napačen format"); continue;
        }
        Koordinati poteza = new Koordinati(x,y);
        if (igra.odigraj(poteza)) return poteza;
        System.out.println(poteza.toString() + " ni možna");
}    }
```

Paketi

S paketi lahko dobro strukturiramo program.

Program `TicTacToe` je razdeljen v treh paketi:

1. Paket `(default package)` z enim razredom `TicTacToe`, ki ima metodo `main`.
2. Paket `vodja` z enim razredom `Vodja`.
3. Paket `logika` z razredi:

   `Igra` `Igralec` `Koordinati` `Polje` `Stanje` `Vrsta`

Razred `Vodja` v paketu `vodja`ki vodi igro. Metode `igramo`, `clovekovaPoteza` in `racunalnikovaPoteza` so statični metode v razredu `Vodja`.

V paketu `logika` je vsa koda, ki implementira stanje in pravila igre.

Dostop

Modifikatori `private`, `protected`, `public` določajo dostop do metodov in polj.

	isti razred	isti paket	podrazredi	povsod
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	
(brez)	✓	✓		
`private`	✓			

Za metodo `metoda()` v razredu `Razred` v paketu `paket`, pišemo:

`metoda()`	v istem razredu
`Razred.metoda()`	v istem paketu
`paket.Razred.metoda()`	izven paketa
`metoda()`	če je: `import paket.Razred`

Razred `Igra` ima konstruktorje, javna polja in metode:

```
Igra()
final int N = 3
Igralec naPotezi()
List<Koordinati> poteze()
Stanje stanje()
boolean odigraj(Koordinati p)
```

Samo `N` je statičen. Ostali polja in metode so dinamični.

```
public class Igra {

    // Velikost igralne plošče je N x N.
    public static final int N = 3;

    // Igralno polje
    private Polje[][] plosca;

    // Igralec, ki je trenutno na potezi.
    private Igralec naPotezi;

    //Nova igra, v začetni poziciji je prazna in na potezi je O.
    public Igra() {
        plosca = new Polje[N][N];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                plosca[i][j] = Polje.PRAZNO;
            }
        }
        naPotezi = Igralec.O;
    }

    public Igralec naPotezi () {
        return naPotezi;
    }
```

```
// @return seznam moznih potez

public List<Koordinati> poteze() {
    LinkedList<Koordinati> ps = new LinkedList<Koordinati>();
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (plosca[i][j] == Polje.PRAZNO) {
                ps.add(new Koordinati(i, j));
            }
        }
    }
    return ps;
}

// @return true, če je bila poteza uspesno odigrana

public boolean odigraj(Koordinati p) {
    if (plosca[p.getX()][p.getY()] == Polje.PRAZNO) {
        plosca[p.getX()][p.getY()] = naPotezi.getPolje();
        naPotezi = naPotezi.nasprotnik();
        return true;
    } else return false;
}
```

```
public Stanje stanje() {
    // Ali imamo zmagovalca?
    Vrsta t = zmagovalnaVrsta();
    if (t != null) {
        switch (plosca[t.x[0]][t.y[0]]) {
        case O: return Stanje.ZMAGA_O;
        case X: return Stanje.ZMAGA_X;
        case PRAZNO: assert false;
        }
    }
    // Ali imamo kakšno prazno polje?
    // Ce ga imamo, igre ni konec
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (plosca[i][j] == Polje.PRAZNO) return Stanje.V_TEKU;
        }
    }
    // Polje je polno, rezultat je neodlocen
    return Stanje.NEODLOCENO;
}
```

```
// Objekt, ki predstavlja eno vrsto na plošči.

class Vrsta {
    public int[] x; public int[] y;
    public Vrsta(int[] x, int[] y) { this.x = x; this.y = y; }
}

private static final List<Vrsta> VRSTE = new LinkedList<Vrsta>();

// Iniciraliziramo N-vrste
static { // Ta koda se izvede enkrat na zacetku
    int[][] smer = {{1,0}, {0,1}, {1,1}, {1,-1}};
    for (int x = 0; x < N; x++) {
        for (int y = 0; y < N; y++) {
            for (int[] s : smer) {
                int dx = s[0]; int dy = s[1];
                if ((0 <= x + (N-1) * dx) && (x + (N-1) * dx < N) &&
                        (0 <= y + (N-1) * dy) && (y + (N-1) * dy < N)) {
                    int[] vrsta_x = new int[N];
                    int[] vrsta_y = new int[N];
                    for (int k = 0; k < N; k++) {
                        vrsta_x[k] = x + dx * k;
                        vrsta_y[k] = y + dy * k;
                    }
                    VRSTE.add(new Vrsta(vrsta_x, vrsta_y));
                }
        ...
}
```

```
// return igralec, ki ima zapolnjeno vrsto @{t}, ali {@null}, ce nihče
private Igralec cigavaVrsta(Vrsta t) {
    int count_X = 0;
    int count_O = 0;
    for (int k = 0; k < N && (count_X == 0 || count_O == 0); k++) {
        switch (plosca[t.x[k]][t.y[k]]) {
        case O: count_O += 1; break;
        case X: count_X += 1; break;
        case PRAZNO: break;
        }
    }
    if (count_O == N) { return Igralec.O; }
    else if (count_X == N) { return Igralec.X; }
    else { return null; }
}

// return zmagovalna vrsta, ali {@null}, če je ni
public Vrsta zmagovalnaVrsta() {
    for (Vrsta t : VRSTE) {
        Igralec lastnik = cigavaVrsta(t);
        if (lastnik != null) return t;
    }
    return null;
}
```