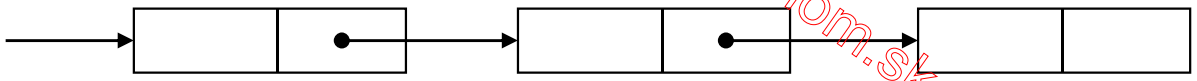
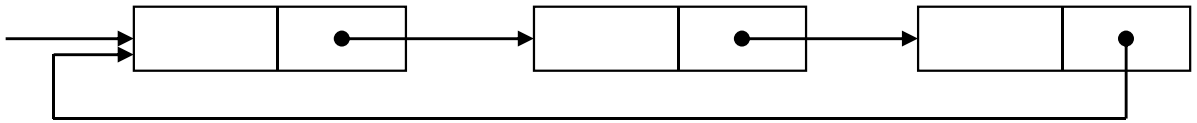
Spojové struktury

- Spojová struktura (linked structure):
  - množina objektů propojených pomocí spojů (odkazů, referencí, ukazatelů)
- Spoj často vyjadřuje vztah předchůdce – následník
- Lineární spojové struktury (spojové seznamy): každý prvek struktury má nanejvýš jednoho následníka
- Příklady spojových seznamů:
  - jednosměrný spojový seznam



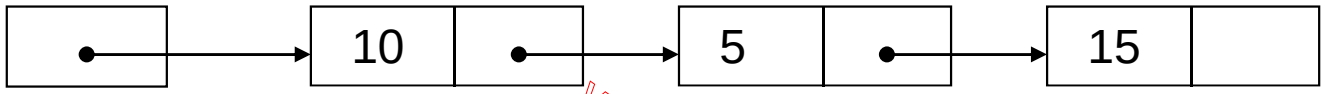
  - dvousměrný spojový seznam

  - cyklický (jednosměrný) spojový seznam



Spojové seznamy

- Jednosměrné spojové seznamy jsme použili pro implementaci zásobníku a fronty
- Jiný příklad: spojový seznam čísel typu *int*



- Seznam vytvoříme z objektů třídy *Prvek*

```
class Prvek {
  int hodn;
  Prvek dalsi;
  public Prvek(int h, Prvek d) {
    hodn = h; dalsi = d;
  }

  public int hodn() {
    return hodn;
  }

  public Prvek dalsi() {
    return dalsi;
  }
}
```

Příklad použití spojového seznamu

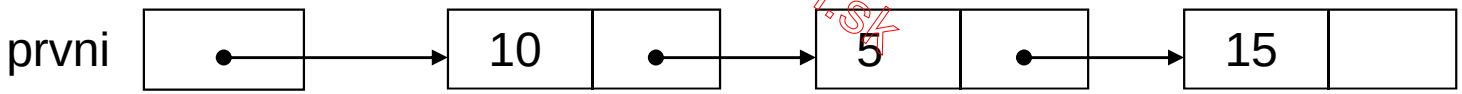
- Příklad: přečíst řadu čísel zakončenou nulou a vypsat čísla v opačném pořadí
- Nástin řešení:
  - z přečtených čísel budeme vytvářet spojový seznam, jehož první prvek bude referencovat proměnná *prvni* typu *Prvek*
  - na počátku je seznam prázdný

```
Prvek prvni = null;
```

  - přečtené číslo vložíme na začátek spojového seznamu

```
prvni = new Prvek(cislo, prvni);
```

  - příklad seznamu po přečtení čísel 15, 5 a 10



  - po přečtení všech čísel bude seznam obsahovat přečtená čísla v opačném pořadí, stačí jej projít a čísla vypsat

```
Prvek pom = prvni;
while (pom!=null) {
  Sys.p(pom.hodn()+" "); pom = pom.dalsi();
}
```

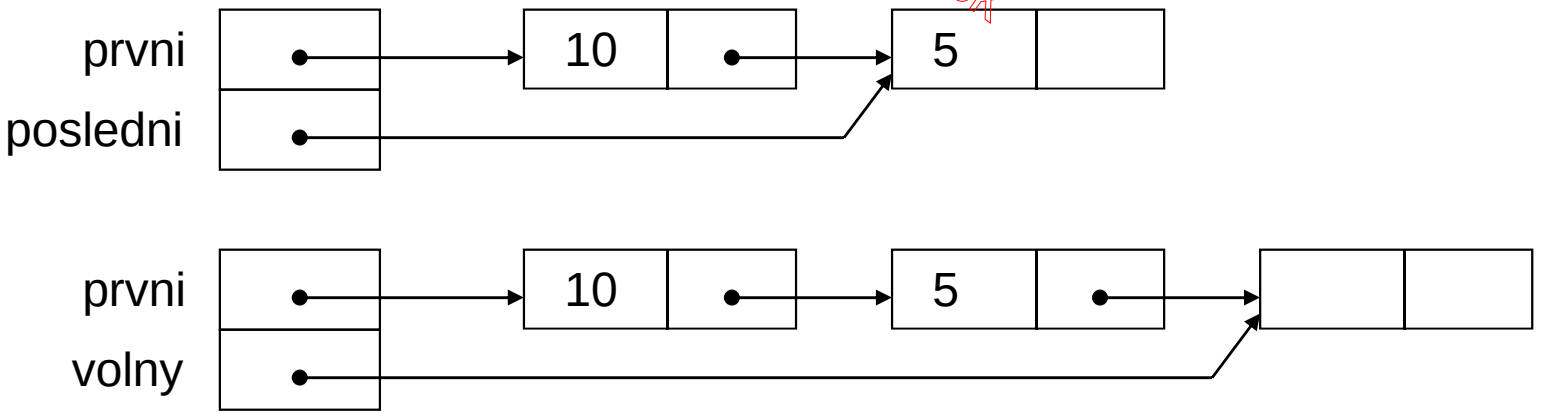
Příklad použití spojového seznamu

- Výsledné řešení:

```
public class ObraceniCisel {
  public static void main(String[] args) {
    Sys.pln("zadejte řadu čísel zakončených nulou");
    Prvek prvni = null;
    int cislo = Sys.readInt();
    while (cislo!=0) {
      prvni = new Prvek(cislo, prvni);
      cislo = Sys.readInt();
    }
    Sys.pln("čísla v opačném pořadí");
    Prvek pom = prvni;
    while (pom!=null) {
      Sys.p(pom.hodn()+" ");
      pom = pom.dalsi();
    }
    Sys.pln();
  }
}
```

Další operace se spojovým seznamem

- Napišme třídu reprezentující spojový seznam celých čísel s těmito operacemi:
  - vložení čísla na začátek seznamu
  - vložení čísla na konec seznamu
  - test, zda číslo je v seznamu
  - výpis čísel uložených v seznamu
- Vložení prvku na konec spojového seznamu identifikovaného pouze odkazem na první prvek vyžaduje najít poslední prvek (lineární složitost)
- Vložení prvku na konec seznamu bude mít konstantní složitost, použijeme-li jednu z následujících reprezentací:



Třída SeznamCisel

- Vložení na konec seznamu bude jednodušší, použijeme-li reprezentaci s odkazem na volný prvek (proč?)

```
public class SeznamCisel {
  Prvek prvni;
  Prvek volny;

  public SeznamCisel() {
    prvni = new Prvek();
    volny = prvni;
  }

  public void vlozNaZacatek(int x) {
    prvni = new Prvek(x, prvni);
  }

  public void vlozNaKonec(int x) {
    volny.hodn = x;
    volny.dalsi = new Prvek();
    volny = volny.dalsi;
  }
```

Třída SeznamCisel

```
  public boolean jePrvkem(int x) {
    volny.hodn = x;
    Prvek pom = prvni;
    while (pom.hodn!=x) pom = pom.dalsi;
    return pom!=volny;
  }

  public void vypis() {
    Prvek pom = prvni;
    while (pom!=volny) {
      Sys.p(pom.hodn+" ");
      pom = pom.dalsi;
    }
    Sys.pln();
  }
}
```


- Poznámka: řešení pomocí odkazu na poslední prvek je vloženo jako komentář do třídy *SeznamCisel*

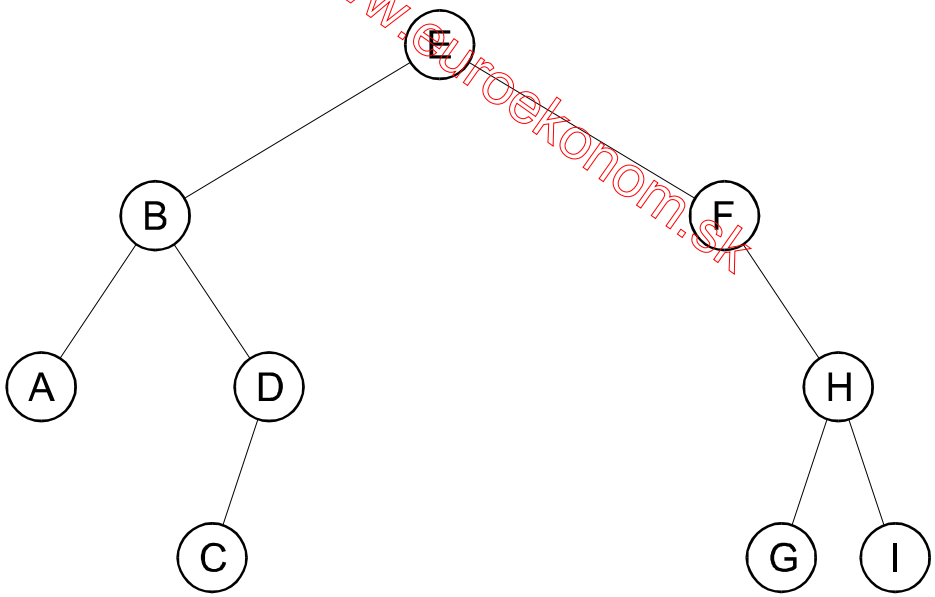
Použití třídy SeznamCisel

- Program, který přečte řadu čísel zakončených nulou a vypíše:
  - přečtená čísla v opačném pořadí
  - seznam různých čísel

```
public static void main(String[] args) {
   SeznamCisel obracena = new SeznamCisel();
   SeznamCisel ruzna = new SeznamCisel();
   Sys.pln("zadejte řadu čísel zakončenou nulou");
   int x = Sys.readInt();
   while (x!=0) {
     obracena.vlozNaZacatek(x);
     if (!ruzna.jePrvkem(x))
       ruzna.vlozNaKonec(x);
     x = Sys.readInt();
   }
   Sys.pln("čísla v opačném pořadí");
   obracena.vypis();
   Sys.pln("seznam různých čísel");
   ruzna.vypis();
 }
```

Stromy

- Lineární spojová struktura (spojový seznam)
  - každý prvek má nanejvýš jednoho následníka
- Nelineární spojová struktura (strom):
  - každý prvek může mít více následníků
- Binární strom: každý prvek (uzel) má nanejvýš dva následníky



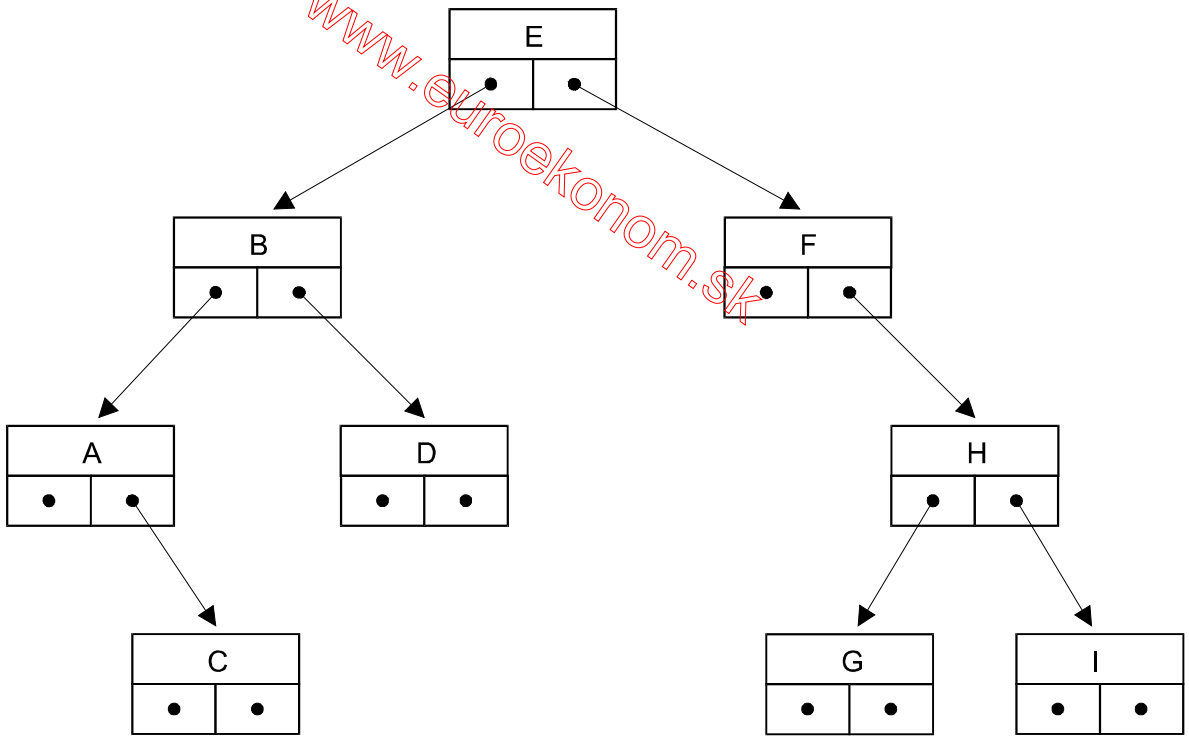
- Některé pojmy: kořen stromu, levý podstrom, pravý podstrom, list

Realizace binárního stromu

- Třída pro realizaci:

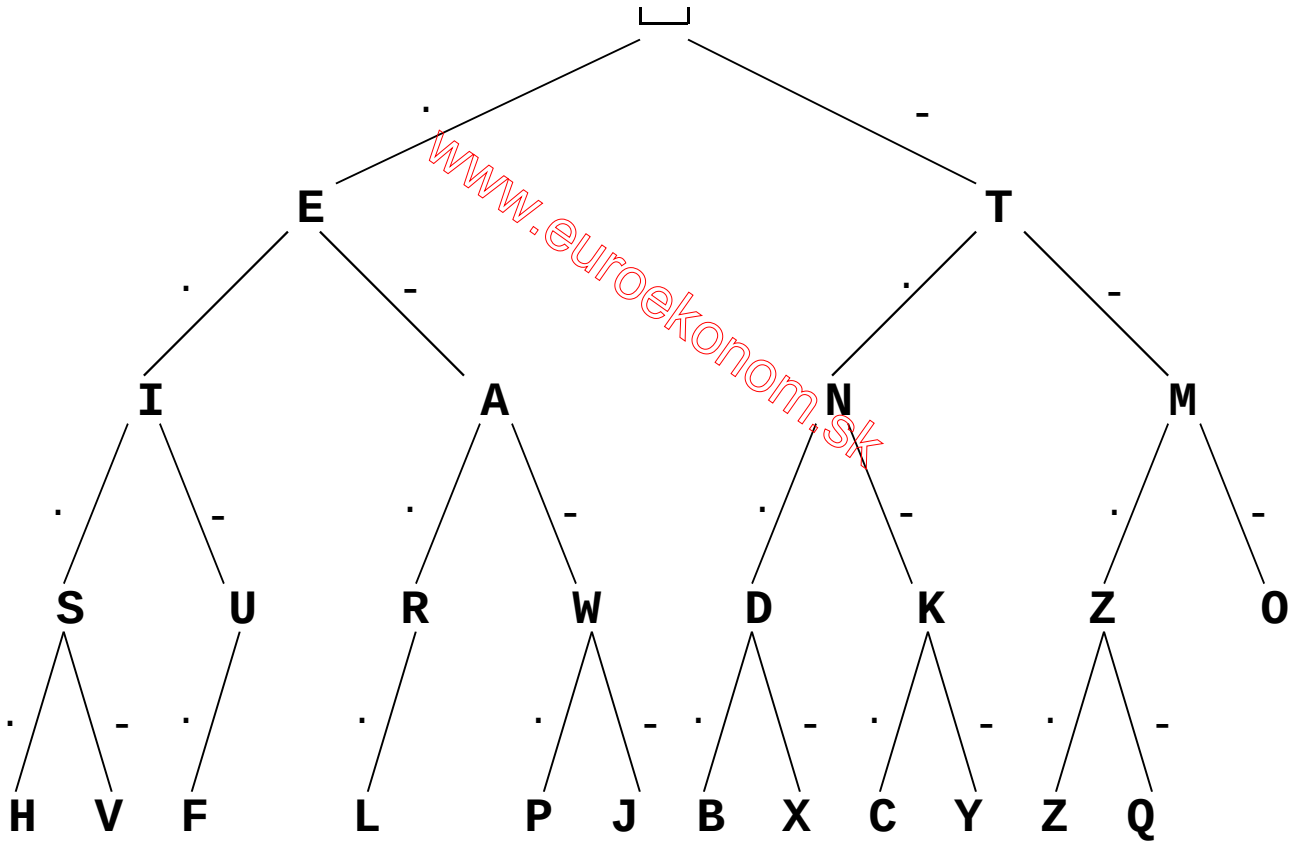
```
class Uzel {
  char hodn;
  Uzel levy, pravy;
  ...
```

- Příklad binárního stromu:



Příklad – dekódování morseovky

- Pro dekódování textu zapsaného v Morseově abecedě lze použít následující binární strom



Příklad – dekódování morseovky

- Strom vytvoříme z objektů typu *MUzel*

```
class MUzel {
  char znak;
  MUzel tecka, carka;

  public MUzel(char z) {
    znak = z; tecka = null; carka = null;
  }

  public MUzel(char z, MUzel t, MUzel c) {
    znak = z; tecka = t; carka = c;
  }
}
```

Příklad – dekódování morseovky

- Pro vytvoření stromu zavedeme funkci:

```
static MUzel strom() {
  return
    new MUzel(' ',
      new MUzel('E',          // .
        new MUzel('I',         // ..
          new MUzel('S',        // ...
            new MUzel('H'),     // ....
            new MUzel('V')      // ...-
          ),
          new MUzel('U',        // ..-
            new MUzel('F'),     // ..-.
            null                // ..—
          )
        ),
        new MUzel('A',         // .-
          ...
        )
      ),
      new MUzel('T',          // -
        ...
      )
    );
  }
```

Příklad – dekódování morseovky

- Dekódování zapíšeme jako funkci, jejímž parametrem je řetězec obsahující Morseův kód a výsledkem je řetězec tvořený odpovídajícími znaky latinské abecedy

```
static String dekoduj(String s) {
   MUzel aktualni = koren;
   String vysl = "";
   for (int i=0; i<s.length(); i++) {
     char z = s.charAt(i);
     if (aktualni!=null)
       if (z=='.') aktualni = aktualni.tecka;
       else if (z=='-') aktualni = aktualni.carka;
       else {
         vysl = vysl + aktualni.znak;
         aktualni = koren;
       }
     else {
       vysl = vysl + '?';
       aktualni = koren;
     }
   }
   return vysl;
 }
```

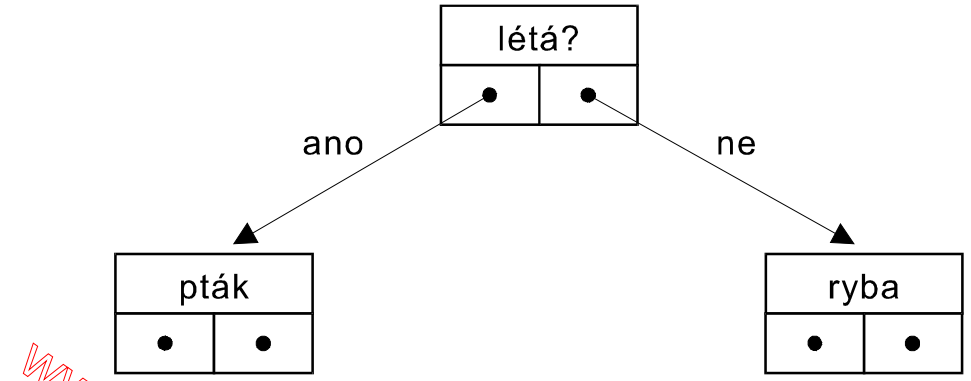
Příklad - hra „Jaké zvíře si myslíš“

- Příklad dialogu:

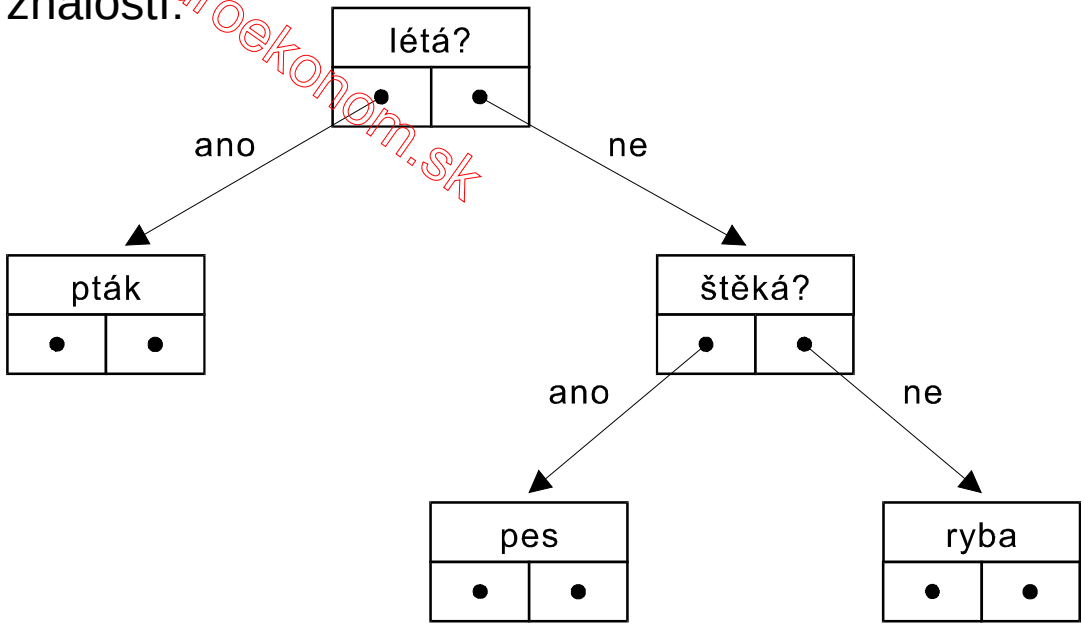
```
Myslíte si nějaké zvíře?
Ano
Zvíře, které si myslíte létá?
Ne
Je to ryba?
Ne
Dám se podat. Jaké zvíře jste myslel?
Pes
Napište otázku vystihující rozdíl mezi pes a ryba!
štěká?
Pro zvíře, které jste myslel, je odpověď ano či ne?
ano
Dekuji.
Chcete hrát ještě jednou?
Ano
Myslíte si nějaké zvíře?
Ano
...
```

Příklad - hra „Jaké zvíře si myslíš“

- Počáteční strom dialogu :



- Strom dialogu po doplnění znalostí:



Příklad - hra „Jaké zvíře si myslíš“

- Hrubé řešení:

```
"úvod dialogu";
"aktuálním uzlem je kořen stromu";
do {
  "polož otázku uvedenou v aktuálním uzlu";
  if ("odpověď je ano")
    "aktuálním uzlem je levý následník"
  else
    "aktuálním uzlem je pravý následník"
} while ("aktuální uzel není list");
"polož závěrečnou otázku, název zvířete vyber z
 aktuálního uzlu";
if ("odpověď je ano")
  "hádání bylo úspěšné"
else {
  "hádání bylo neúspěšné"; "doplň znalosti"
}
```

Příklad - hra „Jaké zvíře si myslíš“

- Podrobné řešení
- Třída uzlů stromu:

```
class Uzel {
  String text;
  Uzel ano, ne;

  public Uzel(String t) {
    text = t; ano = null; ne = null;
  }

  public Uzel(String t, Uzel a, Uzel n) {
    text = t; ano = a; ne = n;
  }

  public boolean jeList() {
    return ano==null && ne==null;
  }
}
```

Příklad - hra „Jaké zvíře si myslíš“

- Hlavní funkce:

```
public class Hra {
  public static void main(String[] args) {
    Uzel koren = inicializaceStromu();
    for (;;) {
      Sys.pln("Myslíte si nějaké zvíře?");
      if (!odpovedAno()) break;
      Uzel aktualni = koren;
      do {
        Sys.pln(aktualni.text);
        if (odpovedAno()) aktualni = aktualni.ano;
        else aktualni = aktualni.ne;
      } while (!aktualni.jeList());
      Sys.pln("Je to "+aktualni.text+"?");
      if (odpovedAno()) Sys.pln("Uhádl jsem");
      else {
        Sys.pln("Neuhádl jsem. Prosím o doplnění znalostí");
        doplnPodstrom(aktualni);
      }
      Sys.pln("Děkuji. Chcete pokračovat?");
      if (!odpovedAno()) break;
    }
  }
```

Příklad - hra „Jaké zvíře si myslíš“

- Pomocné funkce:

```
static boolean odpovedAno() {
  String s = Sys.readLine();
  if (s.length()>0 &&
      (s.charAt(0)=='a' || s.charAt(0)=='A'))
      return true;
  else
    return false;
}

static Uzel inicializaceStromu() {
  return new Uzel("létá?",
                  new Uzel("pták", null, null),
                  new Uzel("ryba", null, null));
}
```

Příklad - hra „Jaké zvíře si myslíš“

- Pomocné funkce:

```
static void doplnPodstrom(Uzel p) {
  String noveZvire, novaOtazka;
  Uzel novyAno, novyNe;
  Sys.pln("Jaké zvíře jste myslel?");
  noveZvire = Sys.readLine();
  Sys.pln("Napište otázku vystihující rozdíl mezi "+
          noveZvire+" a "+p.text);
  novaOtazka = Sys.readLine();
  Sys.pln("Pro zvíře, které jste myslel, je odpověď ano
           či ne");
  if (odpovedAno()) {
    novyAno = new Uzel(noveZvire);
    novyNe = new Uzel(p.text);
  } else {
    novyAno = new Uzel(p.text);
    novyNe = new Uzel(noveZvire);
  }
  p.text = novaOtazka;
  p.ano = novyAno;
  p.ne = novyNe;
}
```