Objektově orientované programování

Doc. Ing. František Huňka, CSc.

katedra informatiky a počítačů PřF

Ostravská univerzita

frantisek.hunka@osu.cz

Literatura – zdroje informací

- Pecinovský R.: Myslíme objektově v jazyku Java 5.0, Grada 2004
- Barness, D., Kolling, M. Object First with Java. Prentice Hall 2009
- Arlow J., Neustadt I.: UML2 a unifikovaný proces vývoje aplikací. Computer Press 2007
- Gamma E., Helm R., Johnson R., Vlissides J.: Návrh programů pomocí vzorů. Grada 2003
- Pecinovský R.: Návrhové vzory. Computer Press 2007

Literatura – zdroje informací

- Polák, Merunka: Objektově orientované programování, Objektově orientované pojmy. Softwarové noviny 1993 série článků.
- www.oracle.com - prostředí jazyka Java

Obsah kurzu 1/3

1. Úvod do tříd a objektů. Java a její zvláštnosti. UML (Unified Modeling Language) a jeho diagramy.
2. Třídně instanční model. Třída, instance. Zpráva, metoda.
3. Práce s objekty v grafickém prostředí (bod, čára).
4. Skládání objektů, grafické objekty – křížek, obdélník.
5. Skládání objektů – praktická aplikace, přetížené konstruktory.

Obsah kurzu 2/3

6. Práce s poli jako datovými atributy.
7. Případová studie – koruna, účet.
8. Návrhové vzory a jejich použití při návrhu programu. Využití vzoru messanger (přepravka), singleton (jedináček).
9. Balíčky, dědičnost, třída Object.
10. Dědičnost, vztahy mezi nadtřídou a podtřídami.
11. Polymorfismus. Abstraktní třída, rozhraní.
12. Využití polí pro ukládání objektů.

Obsah kurzu 3/3

12. Spojový seznam, využití dědičnosti a skládání (delegování).
13. Stromové struktury.

Obsah přednášky

- Objektově orientované paradigma – perspektiva.
- Objektově orientované jazyky.
- Java jako technologie.
- Formalizace zápisu – třída.
- Unified modeling language – UML.

Objektově orientovaný přístup

- Objektově orientovaný přístup zahrnuje:
  - OO analýzu – co chci dělat,
  - OO návrh (design)
  - implementaci, programování.
- OOP - objektově orientované programování.
- V OOP na rozdíl od procedurálního, logického programování pracujeme pouze s OBJEKTY.
- Pojem „objekt“ je jak v reálném světě, tak v počítačovém světě.

Objektově orientovaný přístup

- Blízkost chápání reálného světa – pojem objekt se vyskytuje jak v reálném světě, tak v „počítačovém světě.
- Obecně mají stejné charakteristiky:
  - vlastnosti & datové atributy (instanční proměnné)
  - schopnosti (funkce) & metody

Pojem OBJEKT

- V objektově orientovaném přístupu je objekt všechno:
  - osoba, student, seznam vykonaných zkoušek, telefonní seznam, seznam dětí, auto, kompilační program, matice, vektor atd.
- Jednotlivé objekty mohou být tvořeny dalšími objekty:
  - auto složeno z: karoserie, 4 kola, volant
  - ubytování složeno z: odkaz na hotel, zákazníka, číslo pokoje, vybavení pokoje

Struktura objektu

- vlastnosti objektu – datové atributy
  datová povaha objektu
- operace (metody), které s objektem můžeme provádět
  funkční povaha objektu

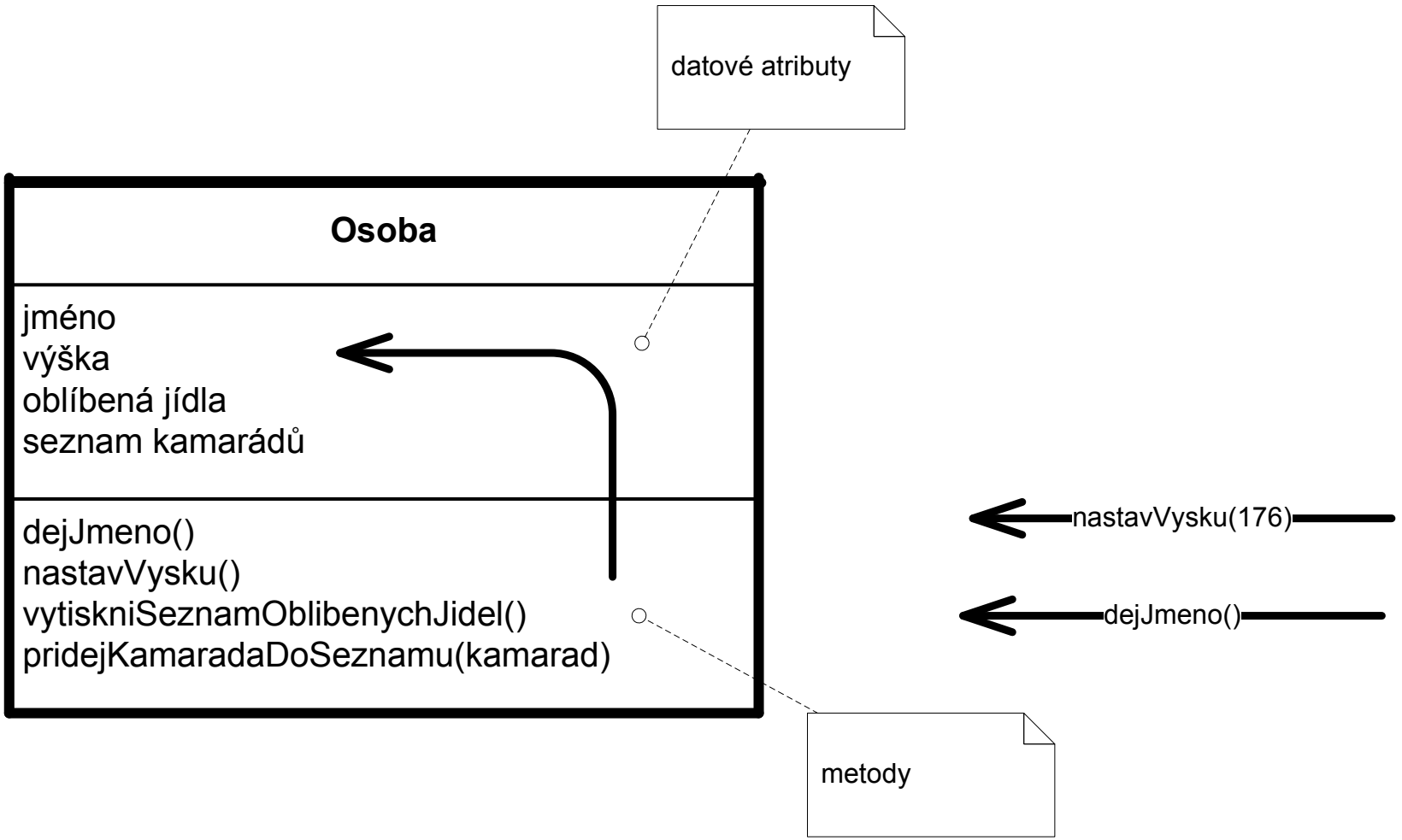
Datová povaha objektu

- **Datová** povaha objektu je dána tím, že objekty se skládají z příslušných vnitřních dat (datových atributů), což jsou buď primitivní typy (real, int, boolean, char), nebo jiné objekty (ze kterých je pak konkrétní objekt složen).

Funkční povaha objektu

- **Funkční** povaha každého objektu je dána tím, že každý objekt má jakoby kolem svých datových atributů (vnitřních dat) obal či zeď, která je tvořena množinou samostatných částí kódu, jež jsou nazývány ***metodami*** a které realizují požadované **operace**.
- **Metody** slouží k tomu, aby umožňovaly konkrétní operace s datovými atributy objektu.
- Metoda má své jméno, deklaruje typy vstupních a výstupních parametrů a obsahuje kód požadovaných dílčích operací.

Zapouzdření objektu



Zapouzdřenost objektu

- **Zapouzdřenost** objektu je důležitá vlastnost OOP a znamená, že s datovými atributy objektu nemůžeme pracovat přímo, ale pouze prostřednictvím deklarovaných **metod** objektu.
- Množina povolených operací s objektem se nazývá ***protokol metod*** objektu

Členění metod

- Metody get/ set – přístupové a modifikační metody.
- Metody vytvářející objekty – konstruktory.
- Ostatní metody.

Pojem třídy

- Většinou, když vytváříme objekty, potřebujeme vytvořit více objektů dané struktury.
- Tyto objekty potřebujeme vytvořit podle nějaké „šablony“ (formy, předpisu).
- Tuto šablonu představuje třída.
- Třídu si můžeme představit jako „továrnu“ na objekty.
- Třída (Class) deklarujeme jedenkrát a vytvoříme libovolný počet objektů (instancí).

Struktura třídy

```
public class Trida {
    // deklarace datovych atributu
      . . .
    // deklarace metod
    . . .
}
```

```
public class Hello {
  // datovy atribut
  private String pozdrav = "Hello World";

  // metoda
  public void go() {
            System.out.println("\n" + pozdrav);
  }
}

1 public class HelloTest {
2   public static void main(String[] args) {
3         // vytvoreni instance (objektu)
4         Hello hello = new Hello();
5         hello.go();
6         }
7 }
```

Poznámky

Třída Hello a

třída HelloTest

```
public class Citac {
  // datovy atribut
  private int pocet;

  // pristupova metoda k atributu pocet
  public int getPocet(){
     return pocet;
  }

  public void pricti(){
     pocet = pocet + 1;
  }
  public void odecti(){
     pocet--;
  }

  public void nuluj(){
     pocet = 0;
  }

  public String toString(){
     return "Citac stav: " + getPocet();
  }

  public void tisk(){
  //this - odkaz na sebe sama
           System.out.println(this.toString());
  }
}
```

Poznámky

Třída Citac

```
 1 public class CitacTest {
 2   public static void main(String[] args) {
 3         Citac ales = new Citac();
 4         Citac eliska = new Citac();
 5         ales.odecti();
 6         ales.odecti();
 7         eliska.pricti();
 8         eliska.pricti();
 9         eliska.odecti();
10         System.out.println("Tisk citace ales: ");
11         ales.tisk();
12         System.out.println("Tisk citace eliska: ");
13         eliska.tisk();
           }
}
```

Poznámky

Třída CitacTest

```
public class Citac {
   private int pocet;

   // pristupova metoda k atributu pocet
   public int getPocet(){
       return pocet;
  }

  // modifikacni metoda atributu pocet
  public void setPocet(int cislo) {
          pocet = cislo;
  }

  public void pricti(){
     //pocet = pocet + 1;
     this.setPocet(this.getPocet() + 1);
  }

  public void odecti(){
      setPocet(getPocet() - 1); //pocet--;
  }

  public void nuluj(){
      this.setPocet(0); //pocet = 0;
  }
  public void prictiCislo(int cislo) {
          setPocet(getPocet() + cislo);
  } }
```

Poznámky

Využití přístupových a modifikačních metod

Metoda toString()

- Metodu deklaruje třída Object.
- Vrací textovou reprezentaci konkrétní třídy (objektu).
- Je nutné ji deklarovat, jinak by se použila metoda toString() třídy Object.

```
 public class CitacTest {
  public static void main(String[] args) {
          Citac ales = new Citac();
          Citac eliska = new Citac();
          ales.setPocet(11);
          eliska.setPocet(3);
          ales.prictiCislo(15);
          ales.odecti();
          eliska.prictiCislo(-24);
          eliska.pricti();
          System.out.println("Tisk citace ales: ");
          ales.tisk();
          System.out.println("Tisk citace eliska: ");
          eliska.tisk();
          }
}
```

Poznámky

Třída CitacTest

Objektově orientovaný přístup

- Způsob nazírání na realitu jako na fyzikální model, který simuluje chování reálné, nebo imaginární části světa;
- Řeší krizi programového vybavení;
- Zahrnuje oblast analýzy, návrhu a implementace programových produktů.

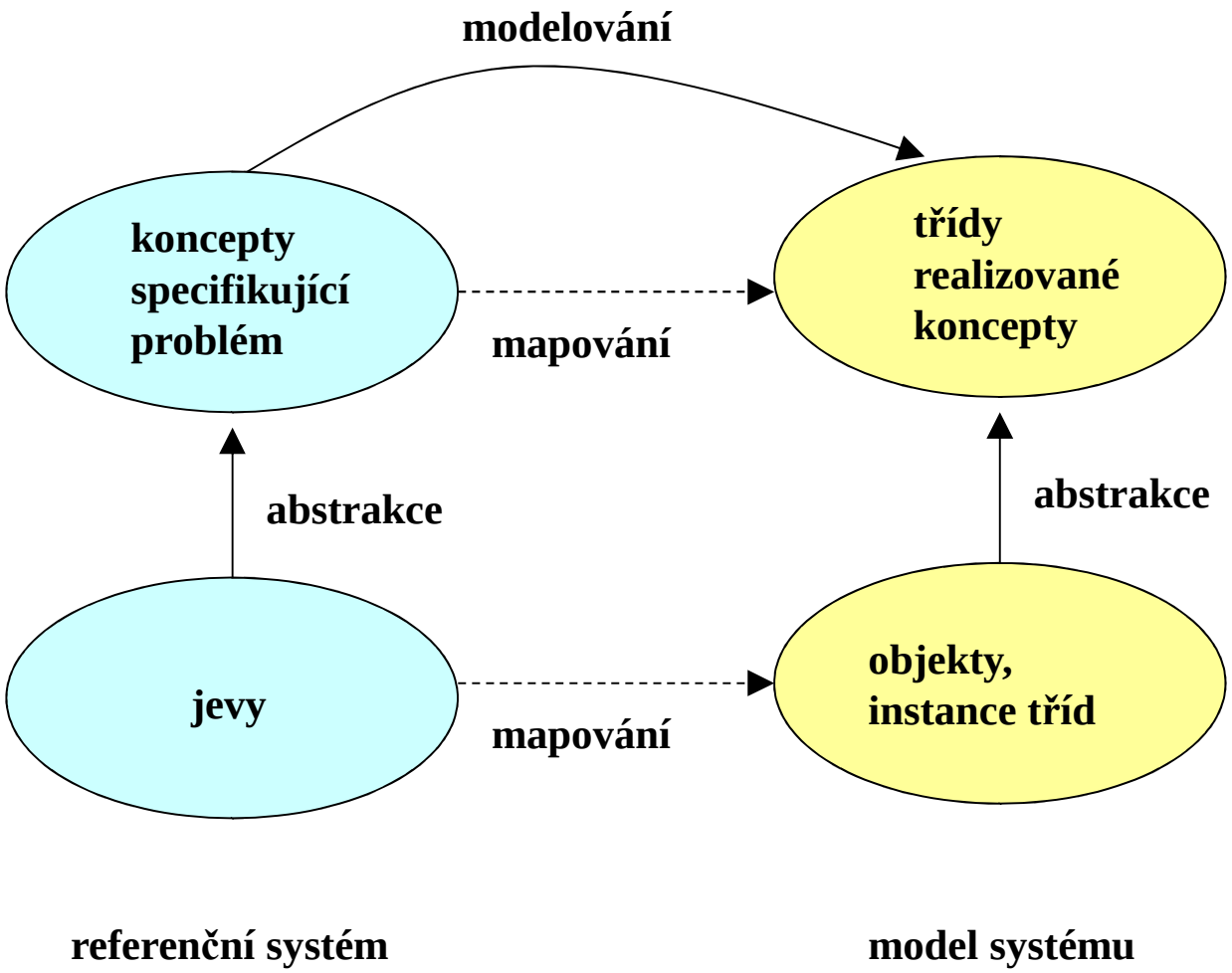
Výhody objektově orientovaného přístupu

- Blízkost chápání reálného světa
- Stabilita návrhu
- Znovupoužitelnost – na úrovni kódu

Výhody objektově orientovaného přístupu 1/3

- Blízkost chápání reálného světa
  - V objektově orientovaném přístupu se vyjadřujeme a pracujeme v pojmech reálného světa a to se neliší od způsobu chápání reálného světa.
    Jev - Koncept
    Objekt – Třída

Programování jako proces modelování



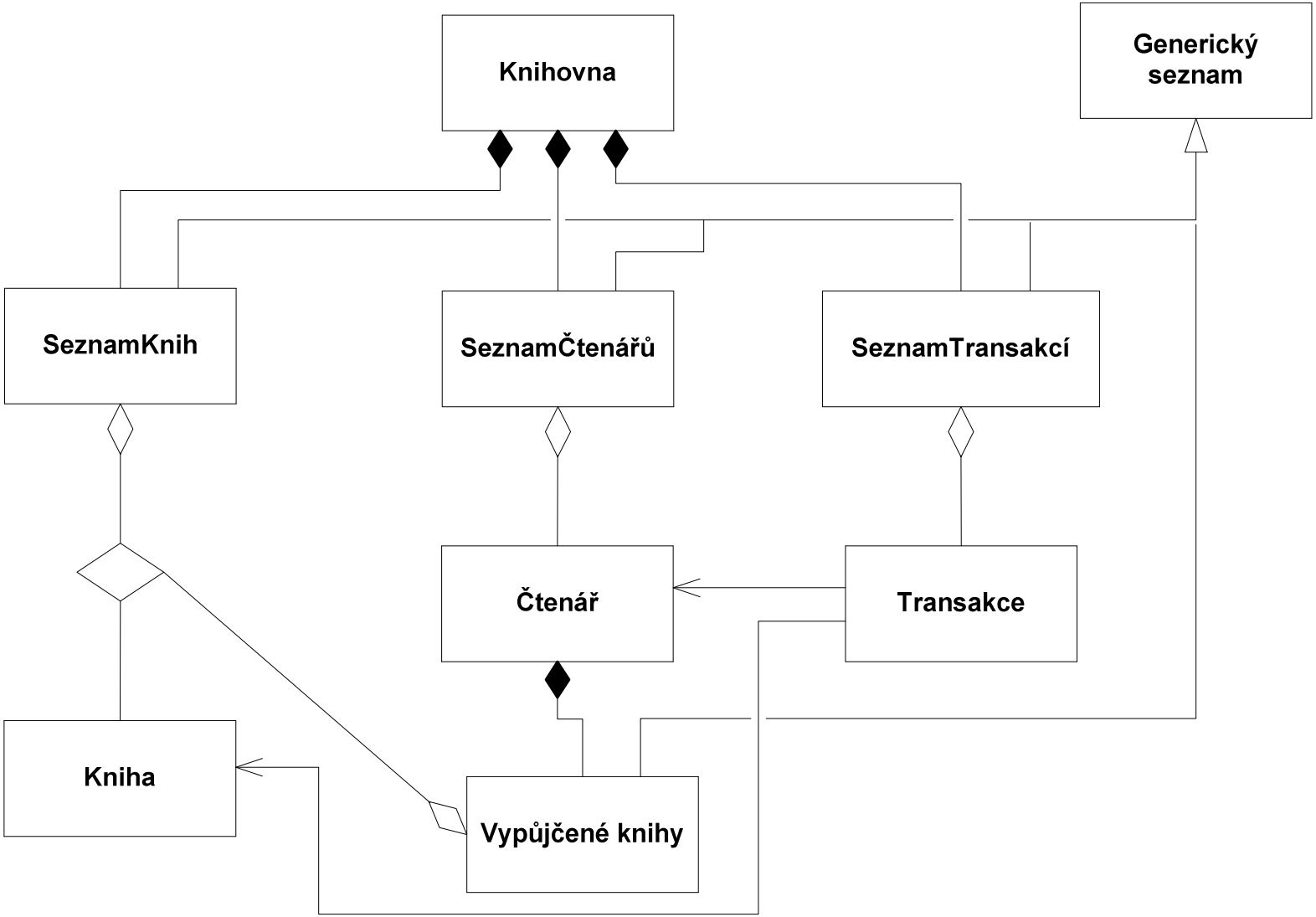
Složky procesu modelování

- Jev je věc, která má danou individuální existenci v reálném světě, nebo v mysli; cokoli reálného.
- Koncept je zevšeobecňující představa kolekce jevů, založena na znalostech společných vlastností jevů v kolekci.

Výhody objektově orientovaného přístupu 2/3

- Stabilita návrhu
  - Místo zaměření se na funkcionalitu systému, je prvním krokem vytvoření fyzikálního modelu reálného světa odpovídající dané aplikaci. Tento model potom vytváří základ pro různé funkce, které systém může mít. Tyto funkce mohou být později měněny a mohou být dodávány nové funkce beze změny základního modelu.

Diagram tříd knihovny



Výhody objektově orientovaného přístupu 3/3

- Znovupoužitelnost – na úrovni kódu.
- Každý OOP systém má knihovnu tříd (hotové třídy).

1. Znovupoužitelnost – vytvoření instance od konkrétní knihovní třídy (nebo námi vytvořené třídy).
2. Vytvoření podtřídy od konkrétní knihovní (naší) třídy a doplnění datových atributů a metod.

Objektově orientované programovací jazyky

- Simula doba vzniku 1967
- Smalltalk model-view-controller
- C++
- Beta
- Self, Ada

Java jako technologie

- Objektově orientovaná
- Nezávislá na platformě
- Robustní, dynamická a bezpečná
- Víceprocesní
- Podporuje vývoj SW pomocí komponent
- Distribuovaná

Java – objektová aplikace pro BlueJ

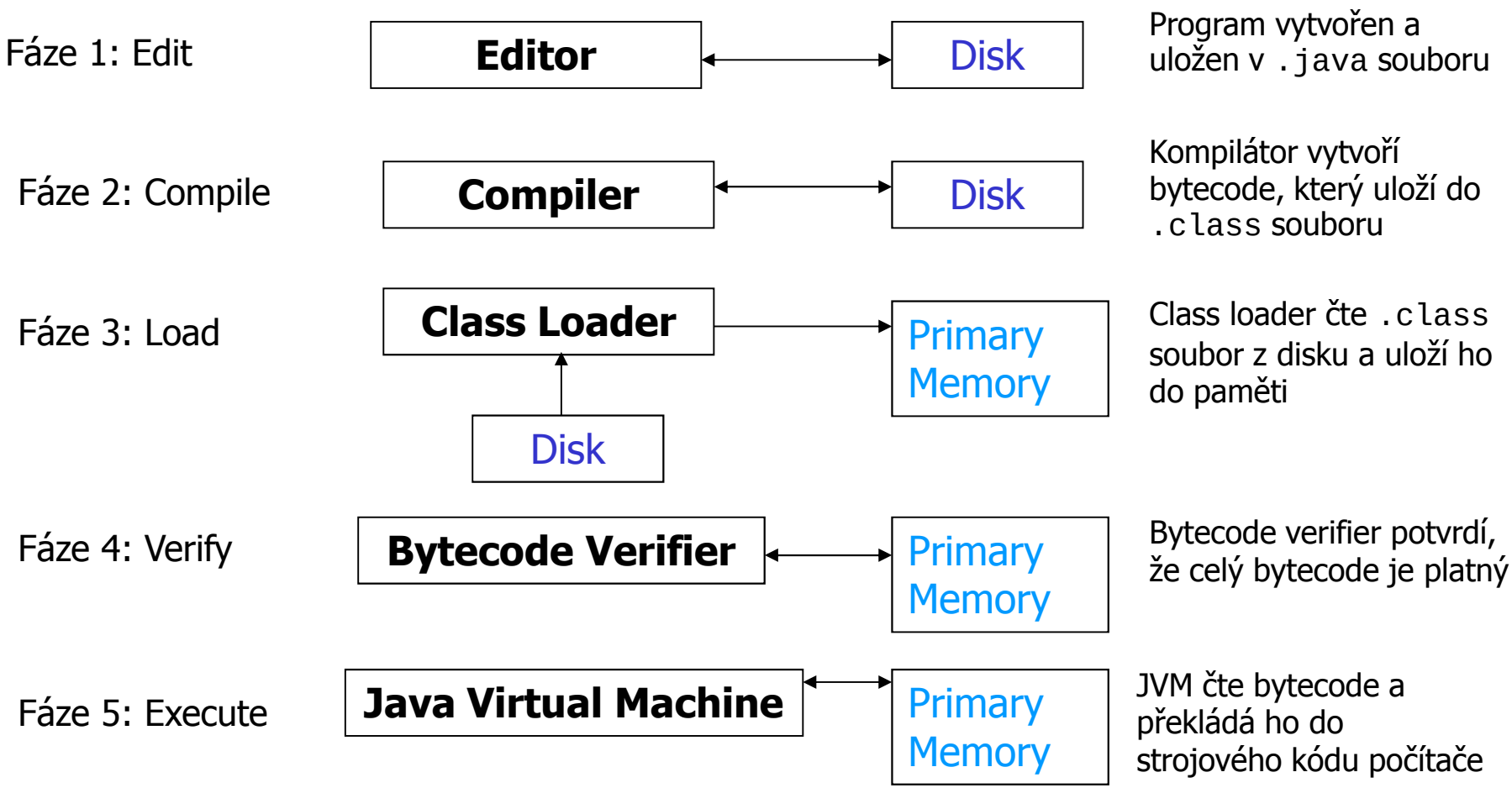
```
// Class Hello:  Hello-world program pro demonstraci v BlueJ

class Hello
{
   // Method that does the work
   public void go()
   {
      System.out.println("Hello, world");
   }
    /**
    *  main method for testing outside BlueJ
    */
   // public static void main(String args[])
   //{  Hello hi = new Hello();
   //    hi.go();   }
}
```

Nezávislost na platformě

- Produktem překladu zdrojových souborů je Java Bytecode (soubory .class) nezávislé na architektuře počítače
  - Knihovny java.awt.*, java.net.*, java.applet.*
- Java Virtual Machine (JVM) – imaginární stroj implementovaný pomocí programové emulace na skutečném stroji
  - Instrukční sada, registry, zásobník (stack), garbage collector

Typické vývojové prostředí v Javě



Programovací jazyk Java

- Java je všeobecně použitelný, konkurentní, silně typový, objektově orientovaný jazyk založený na třídách.
- Normálně kompilovaný do byte-kódové množiny instrukcí a binárního formátu definovaného ve specifikaci Java Virtual Machine.

Programovací jazyk Java

- Java virtual machine je abstraktní počítačový stroj, který má množinu instrukcí a manipuluje pamětí za běhu programu.
- JVM je portovaný na různé platformy k poskytnutí nezávislosti hardwarové a nezávislosti operačního systému.

Struktura objektu – formalizmus zápisu

- Pro deklaraci objektů se v tzv. třídně instančním modelu používá třída (class).
- Základní prvky třídy:
  - název třídy (jedinečný název v rámci balíčku)
  - datové atributy
    - primitivní datové typy
    - objektové datové typy
  - metody (operace i datové atributy vypočítávané při každém přístupu)

Struktura objektu – formalizmus zápisu

- datové atributy:
  - primitivní – int, boolean, double …
  - objektové datové typy – ostatní atributy deklarované pomocí třídy
- metody:
  - zajišťují základní požadované operace (tisk, toString, výpočet …)
  - přístupové a modifikační metody

Zapouzdření objektu (encapsulation)

- Výhody zapouzdření:
- s datovými atributy je možno manipulovat pouze prostřednictvím protokolu deklarovaných metod
- vede k zefektivnění procesu implementace (programování)

```
public class Bod
{
   // deklarace datových atributů
   private int x;
   private int y;

   // deklarace přístupových metod
   public int getX() {
     return x;
   }

   public int getY() {
      return y;
    }

   // deklarace modifikačních metod
   public void setX(int x0) {
      x = x0;
   }

   public void setY(int y) {
     this.y = y;
   }
```

Poznámky

Objektově orientované programování příklady

```
  // deklarace dalších metod
  public String toString()
   {
       String t = "\nX: "+ x +" Y: "+y;
       return t;
   }


  public void tisk() {
       System.out.println("Souradnice bodu "+this.toString());
   }
  // konec deklarace třídy Bod
}
```

Poznámky

Objektově orientované programování příklady

```
public class BodTest {
 public static void main(String args[])   {

   Bod b1 = new Bod(); // vytvoření objektu / instance b1
   Bod b2 = new Bod(); // vytvoření objektu / instance b2
   Bod b3; // kvalifikace proměnné b3
   b3 = new Bod(); // vytvoření objektu / instance b3

  b1.setX(12); b1.setY(-26);
  b1.tisk();

  b2.setX(b1.getY());
  b2.setY(b1.getX());
  String g = b2.toString():
  System.out.println("Bod b2: \n" + g);

  b3.setX(b1.getX + b2.getX());
  b3.setY(b2.getY * b1.getX());
  System.out.println("Bod b3: ");
  b3.tisk();

 }
}
```

Poznámky

Objektově orientované programování příklady

Třída BodTest

Třída bod a atributem jméno

- Pro snadnější identifikaci, přidáme další atribut jméno – typu String

```
public class Bod_Jm
{
   // deklarace datových atributů
   private int x;
   private int y;
   private String jmeno;

   // přístupové metody
   public int getX() {
     return x; }

   public int getY() {
      return y; }

   public String getJmeno() {
       return jmeno; }

   // modifikační metody
   public void setJmeno(String jmeno) {
     this.jmeno = jmeno; }

   public void setX(int x1) {
      x = x1; }

   public void setY(int y) {
     this.y = y; }
```

Poznámky

Třída Bod_Jm

```
  public String toString() {
        String t = "\nNazev bodu: "+jmeno+" X: "+ getX() +" Y: "+ getY();
        return t;
    }

    public void tisk() {
        System.out.println("Souradnice bodu "+this.toString());
    }

   public int soucet() {
     return x + y;
    }

   public double prumer() {
     double v = (double) (x + y) / 2;
     return v;
     // return (double) (x + y) / 2;
   }

}
```

Poznámky

Třída Bod_Jm

```
public class Bod_JmTest {
  public static void main(String args[]) {
     Bod_Jm bod1 = new Bod_Jm();
     Bod_Jm bod2 = new Bod_Jm();

     bod1.setJmeno(“bod_1”);
     bod1.setX(44); bod1.setY(-555);
     bod1.tisk();

     bod2.setJmeno(“muj bod 2”);
     bod2.setY(bod1.getX() + b1.getY());
     bod2.setX(bod1.getY() – b1.getX());
     bod2. tisk();

     System.out.println(“Soucet souradnic bodu bod2: “ + bod2.soucet());

     double prm = bod1.prumer();

   }
 }
```

Poznámky

Třída Bod_JmTest

Co je to UML 1/2

- Unified Modeling Language – (UML) je standardní jazyk pro specifikaci, zobrazení (vizualizaci) vytváření a dokumentaci programových systémů, stejně také pro business modeling a jiné neprogramové systémy.
- UML je nejrozšířenější schéma grafické reprezentace pro modelování objektově orientovaných systémů.

Co je UML 2/2

- UML reprezentuje kolekci nejlepších inženýrských zkušeností, jaké byly úspěšně prověřeny v modelování rozsáhlých složitých systémů.
- UML využívá hlavně grafickou notaci pro vyjádření návrhu programových projektů.
- Používání UML pomáhá projektovým týmům komunikovat, zkoumat potenciální návrhy a prověřovat návrh architektury programovým systémům.

Cíle UML 1/2

1. Poskytnout uživatelům jednoduchý, expresivní vizuální modelovací jazyk, aby mohly vyvíjet a měnit smysluplné modely.
2. Poskytnout mechanismus na rozšíření a další specifikaci základních konceptů.
3. Být nezávislý na konkrétním programovacím jazyku a vytvářených procesech.
4. Poskytnout formální základ pro porozumění jazyku modelování.

Cíle UML 2/2

5. Podpořit vysoce úrovňové rozvojové koncepty jako spolupráce, programové balíčky (frameworks), vzory (patterns) a komponenty.
6. Integrovat nejlepší zkušenosti.

Typy UML diagramů 1/2

- Každý UML diagram je navržen, aby dovolil vývojářům a zákazníkům mít pohled na programový systém z různých perspektiv a z měnících se stupňů abstrakce.
- UML diagramy obecně vytváření vizuální modelovací prostředky, které zahrnují:

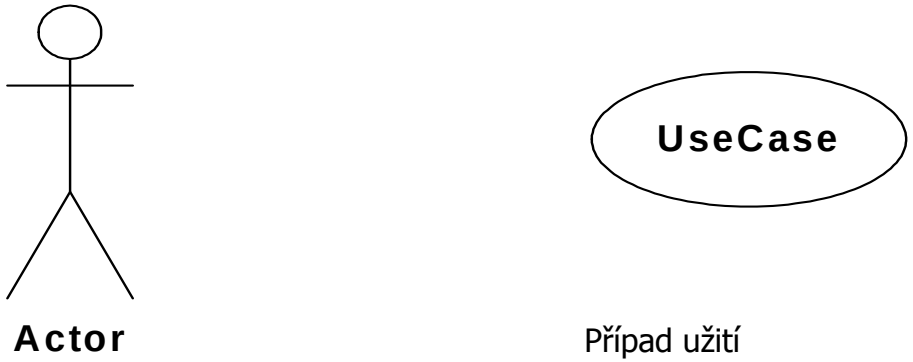
Typy UML diagramů

- Diagram případů užití – use case diagram
- Diagram tříd – class diagram
- Interakční diagramy
  - Sekvenční diagram
  - Diagram spolupráce
- Stavový diagram
- Diagram aktivit
- Fyzické diagramy
  - Diagram komponent
  - Diagram rozmístění – deployment diagram

Diagram případů užití 1/4 –
Use Case Diagram

- Use case (případ užití) je specifikace posloupnosti činností (včetně měnících se a chybových posloupností, které systém může vykonávat prostřednictvím interakce (vzájemného působení) s vnějšími účastníky.
- Případ užití je něco, co účastník (aktor) od systému očekává. Je to „případ užití systému specifickým účastníkem“.
  - Případy užití jsou vždy iniciovány účastníkem.
  - Případy užití jsou vždy napsány z pohledu účastníka.
- Use case diagram (diagram případů užití)– zobrazuje vztah mezi účastníky a případy užití (use cases).
- Use case diagram má dvě hlavní komponenty:
  - Use cases (případy užití)
  - Účastníky - aktory (actors)

Diagram případů užití 2/4



- Účastník - aktor reprezentuje uživatele, nebo jiný systém, který bude v interakci (vzájemném působení) se systémem, který modelujete.
- Use case (případ užití) je externí pohled systému, který reprezentuje nějakou činnost, kterou uživatel může vykonávat, aby dokončil úlohu.

Diagram případů užití 3/4

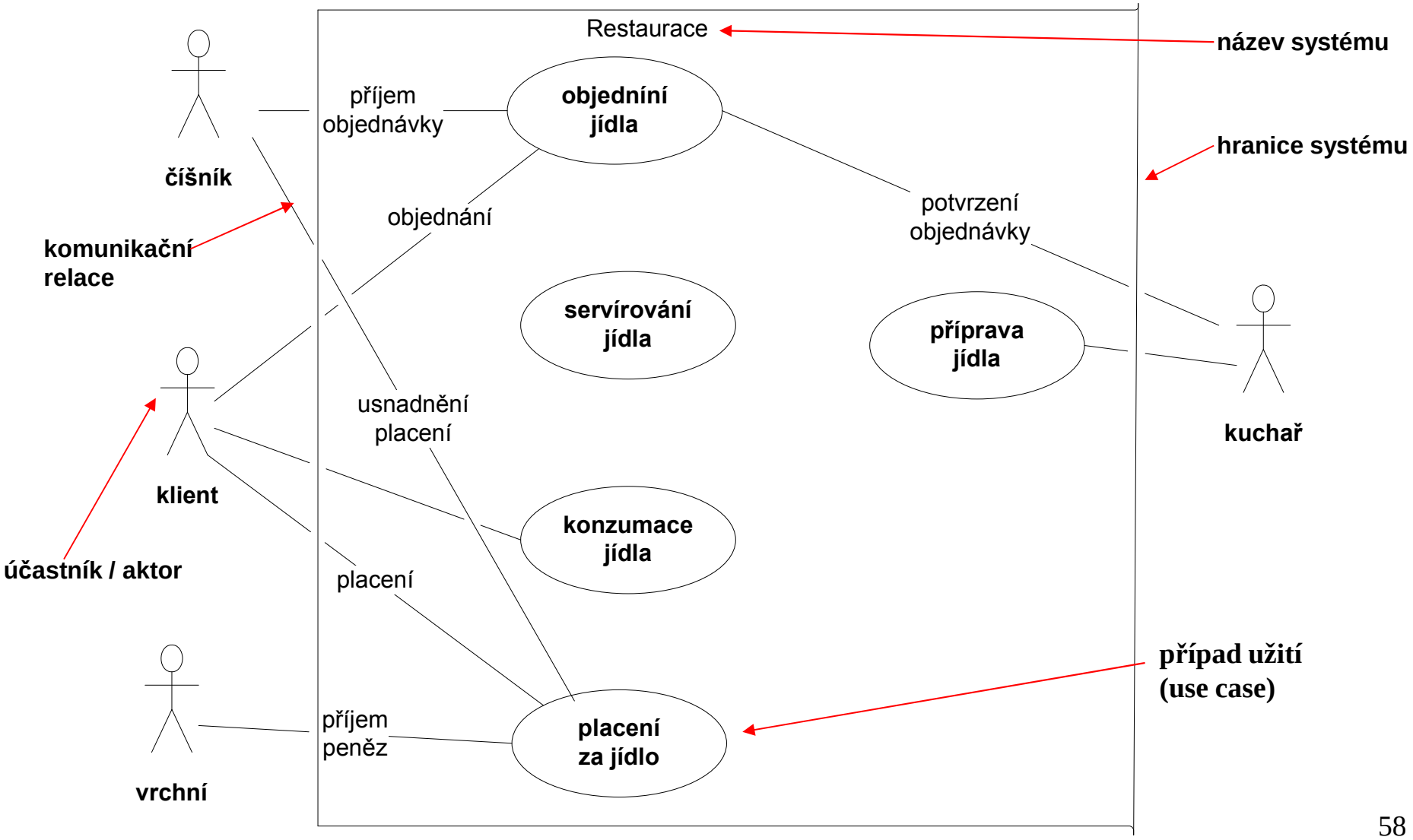


Diagram případů užití – 4/4

- Diagram případu užití může být snadno rozšířen o další činnosti, resp. aktualizovat stávající činnosti.

Diagram tříd 1/2

- Diagram tříd (class diagram) modeluje strukturu a obsah tříd k čemuž používá navržené prvky jako – třídy, pakety a objekty.
- Také zobrazuje vztahy (relace) jako kompozice, dědičnost, asociaci a další.

Diagram tříd

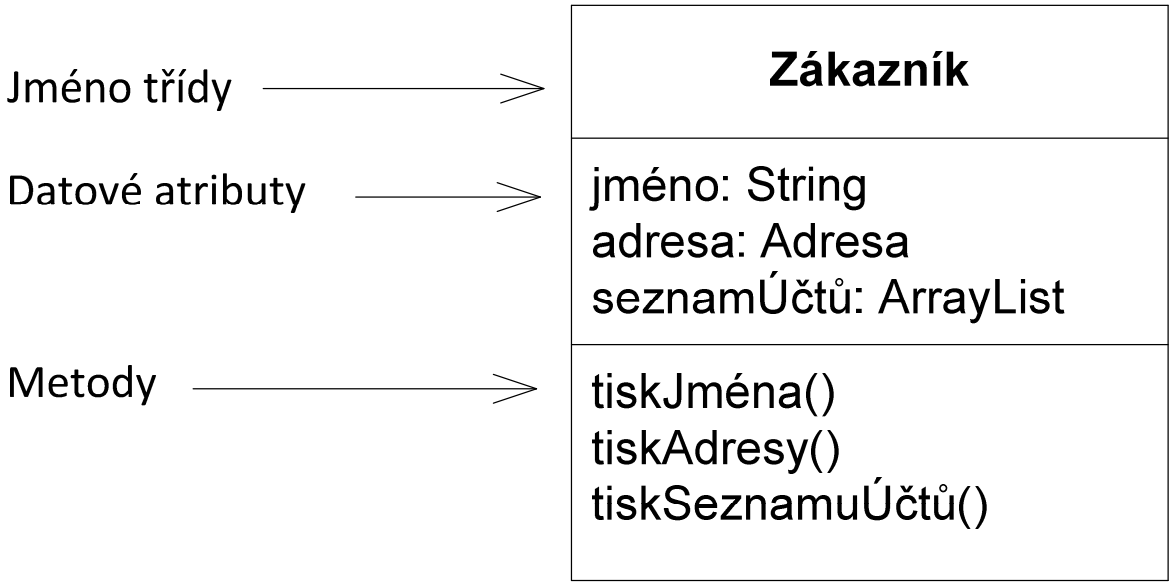
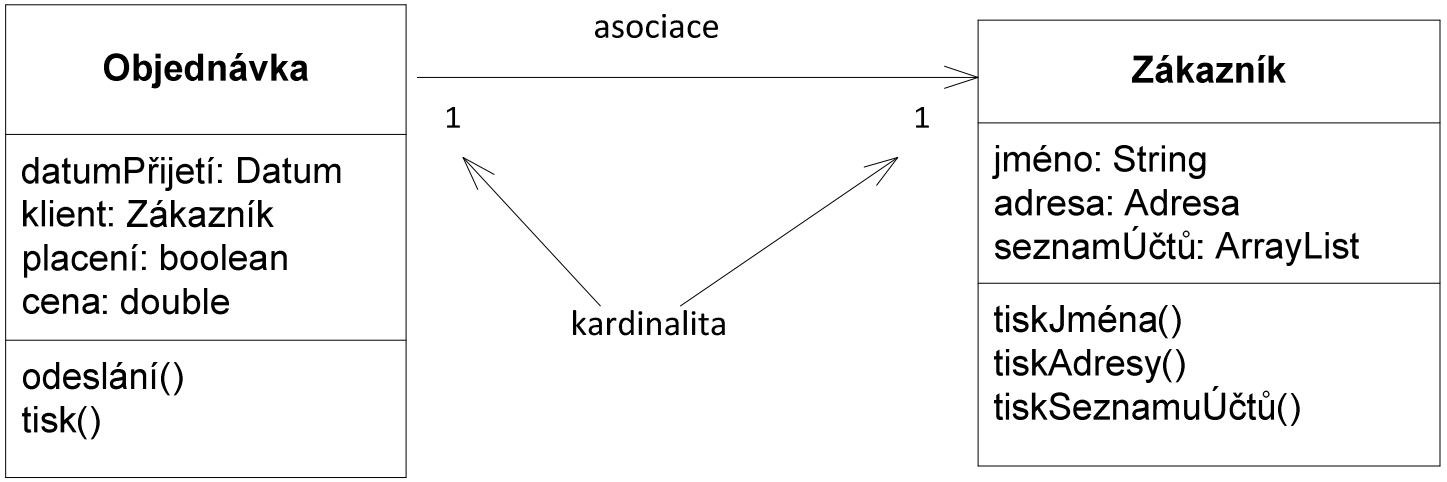
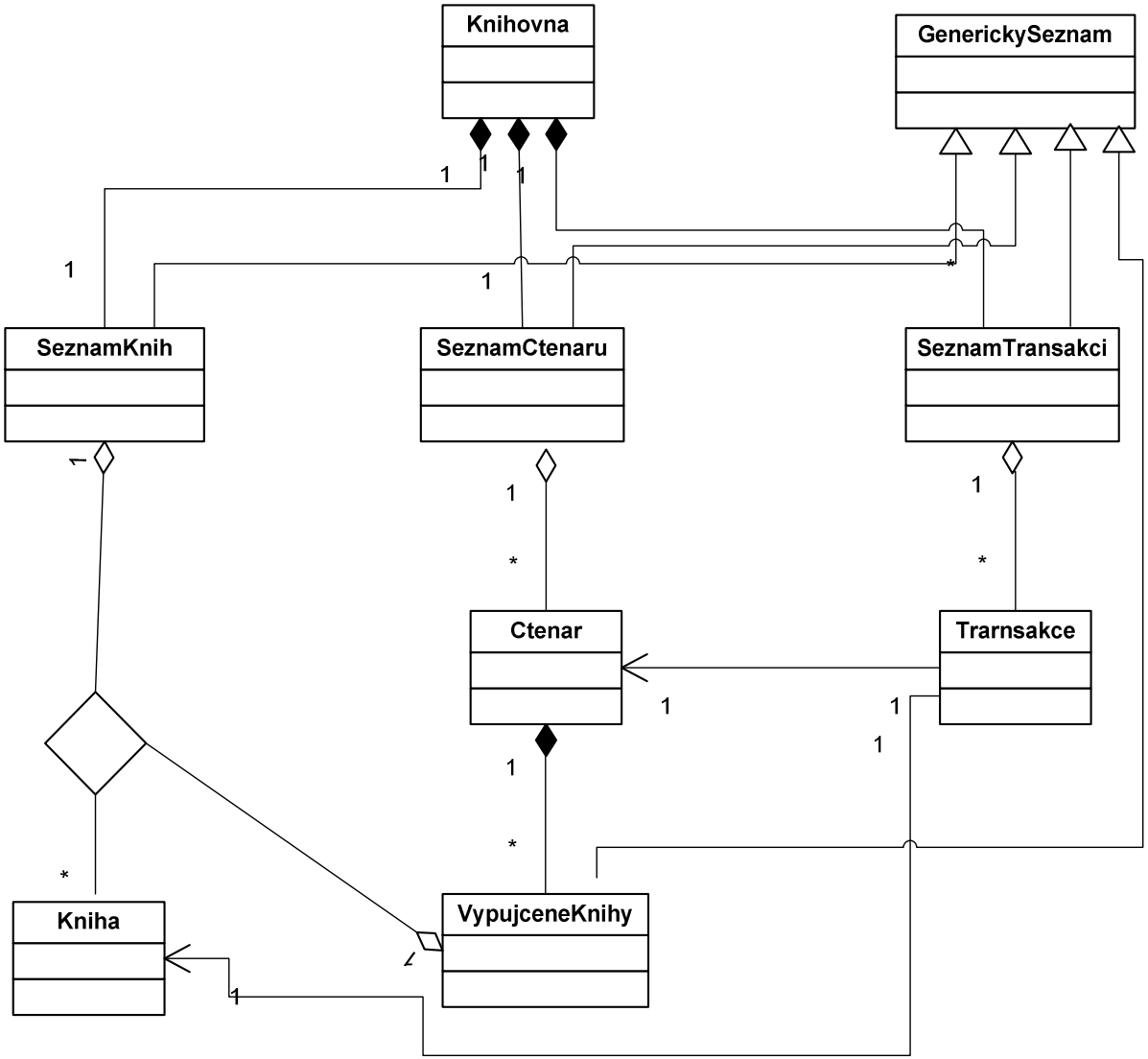


Diagram tříd

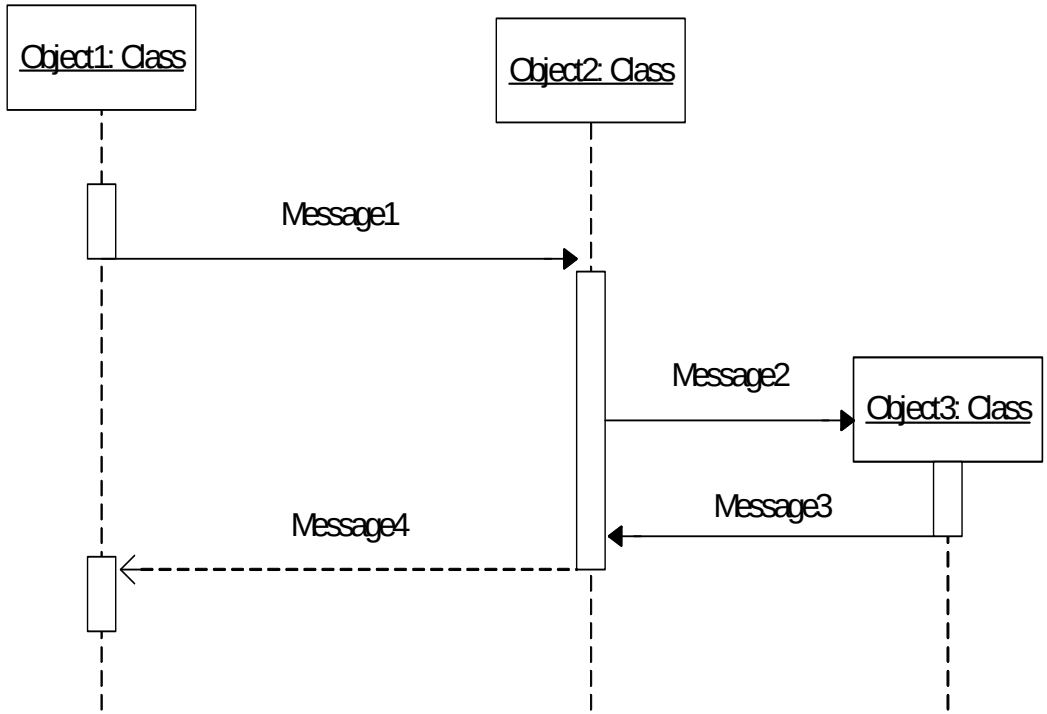


Knihovna – diagram tříd



Interakční diagramy 1/2

- Sequance diagram (sekvenční diagram) – zobrazuje časovou sekvenci objektů účastnící se interakce. Skládá se z vertikální dimenze (čas) a horizontální dimenze (různé objekty).



Interakční diagramy 2/2

- Collaboration diagram (diagram spolupráce) zobrazuje interakce organizované mezi objekty a jejich spojení (vazby) mezi sebou (pořadí zasílaných zpráv mezi objekty).

Stavový diagram – state diagram

- Stavový diagram zobrazuje sekvence stavů, kterými prochází objekt během svého života v závislosti na obdrženém stimulu, spolu s jeho reakcemi a činnostmi.

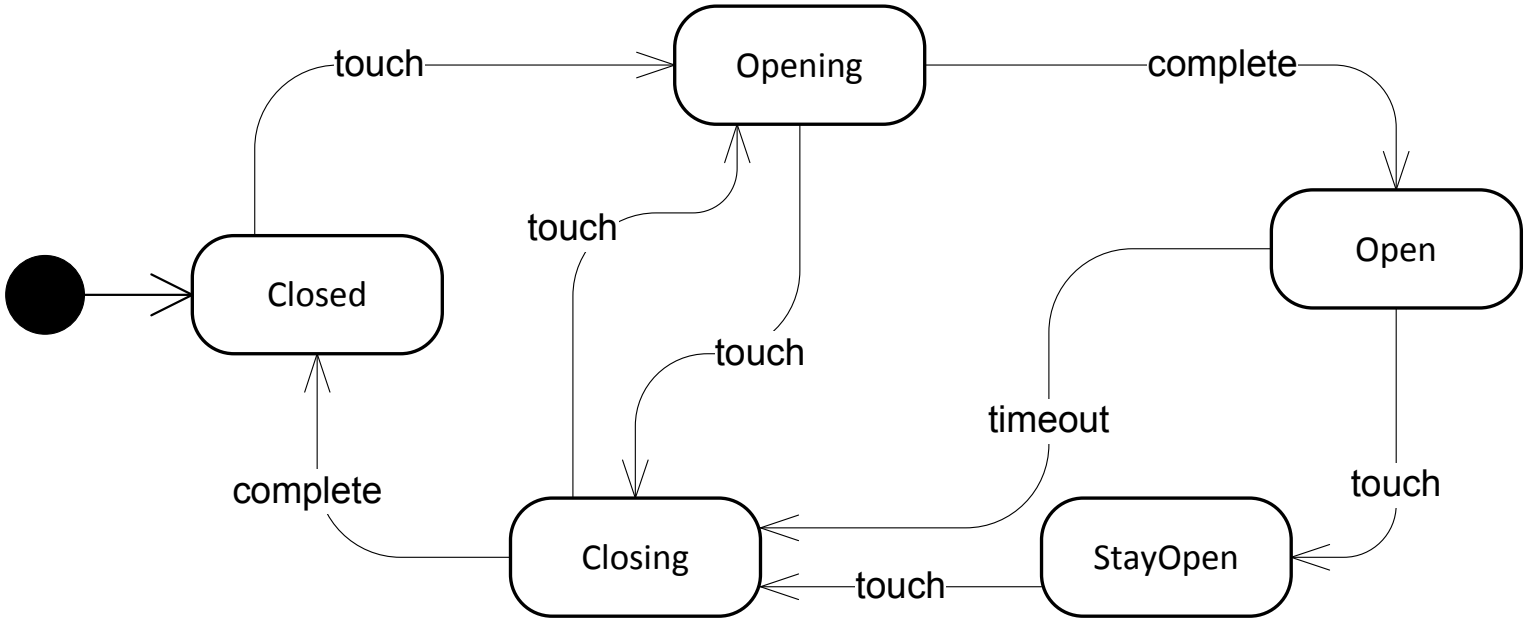
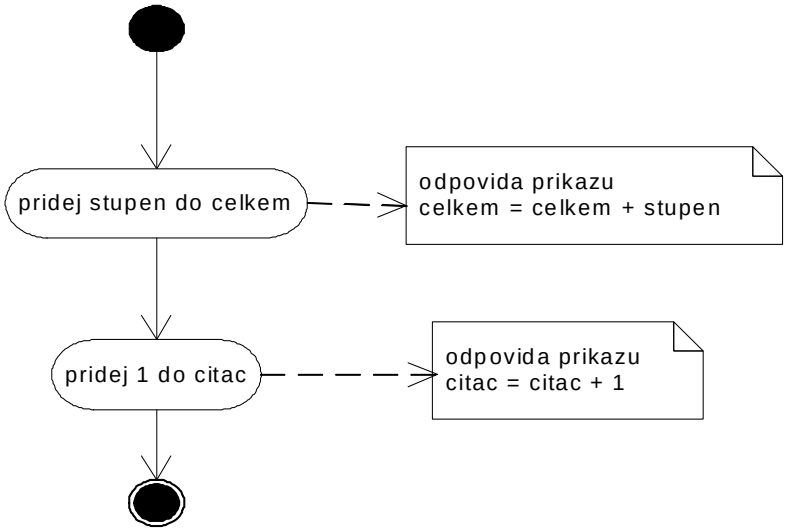


Diagram aktivit

- Zobrazuje speciální stavový diagram, kde většina ze stavů jsou stavy činností a většina přechodů je spouštěna vykonáním akcí ve zdrojových stavech. Tento diagram se zaměřuje na toky řízené vnitřním zpracováním.
- Používá se k zachycení algoritmů v programovacích jazycích – vývojový diagram.

Diagram aktivit - příklady



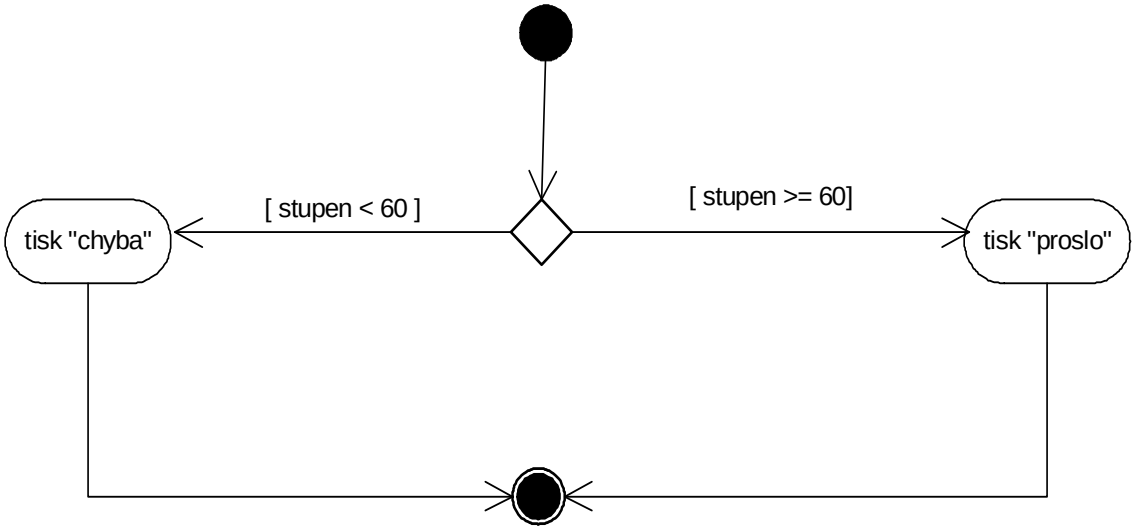
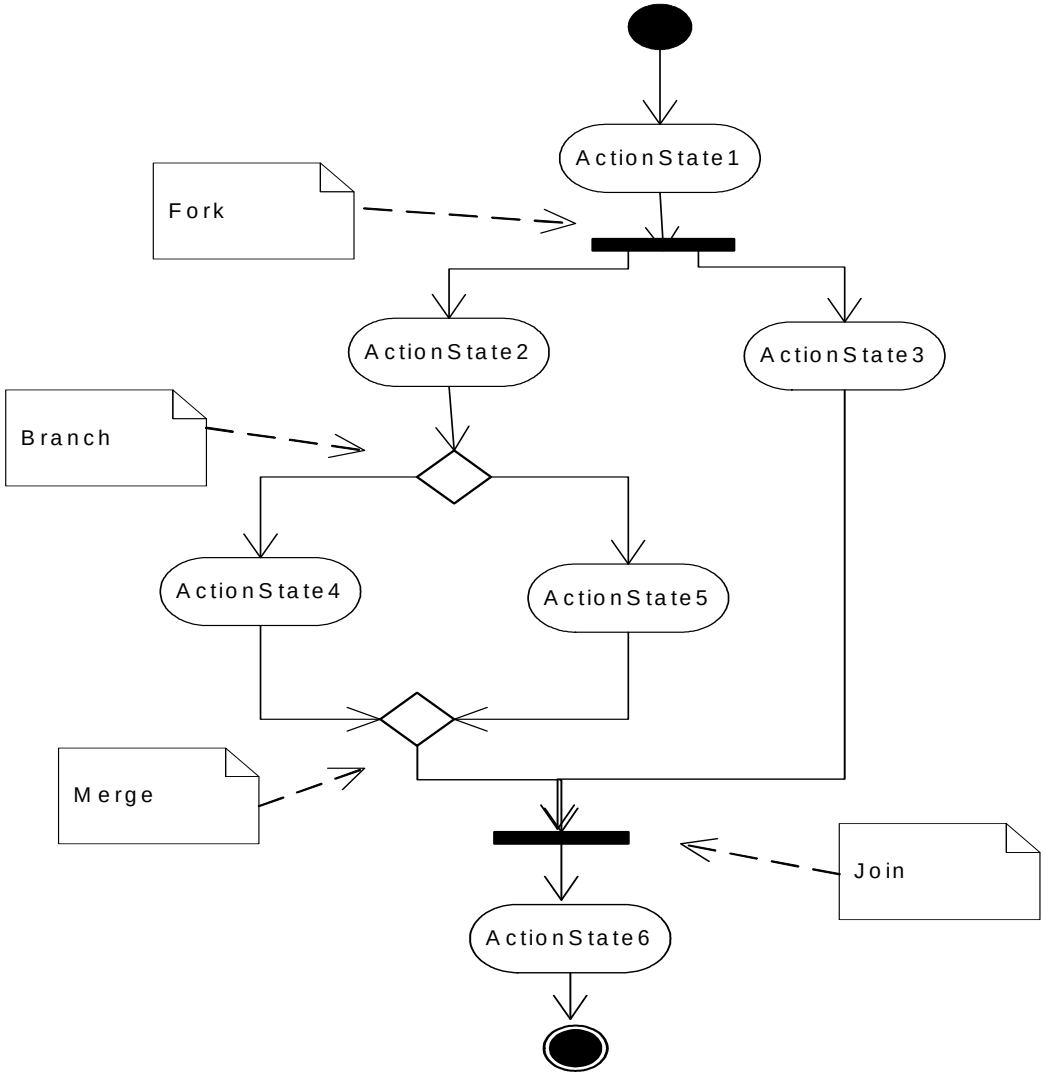


Diagram aktivit - příklady



Fyzické diagramy 1/2

- Diagram komponent– zobrazuje vysokou úroveň paketové struktury samotného kódu. Jsou zobrazeny závislosti mezi komponentami včetně zdrojového kódu komponent, binárního kódu komponent a spustitelné komponenty.

Fyzické diagramy 2/2

- Deployment diagram (diagram nasazení) – zobrazuje konfiguraci prvků běhového zpracování (run-time processing elements) a programových komponent, procesů a na nich žijících objektů.