# Společná část - otázka č. 6

**Principy objektového přístupu, třída jako: programová jednotka, zdroj funkcí, datový typ; struktura objektu, konstruktory, přetěžování, instance třídy, hierarchie tříd, dědění, kompozice; abstraktní třídy, polymorfismus, rozhraní, rozhraní jako typ proměnné, typ interface.**

June 2, 2012

# 1 Objektové programování

Objektově orientované programování (zkracováno na OOP) je metodika vývoje softwaru, založená na následujících myšlenkách, koncepci:

1. Objekty – jednotlivé prvky modelované reality (jak data, tak související funkčnost) jsou v programu seskupeny do entit, nazývaných objekty. Objekty si pamatují svůj stav a navenek poskytují operace (přístupné jako metody pro volání).

2. Abstrakce – programátor, potažmo program, který vytváří, může abstrahovat od některých detailů práce jednotlivých objektů. Každý objekt pracuje jako černá skříňka, která dokáže provádět určené činnosti a komunikovat s okolím, aniž by vyžadovala znalost způsobu, kterým vnitřně pracuje.

3. **Zapouzdření** – zaručuje, že objekt nemůže přímo přistupovat k „vnitřnostem“ jiných objektů, což by mohlo vést k nekonzistenci. Každý objekt navenek zpřístupňuje rozhraní, pomocí kterého (a nijak jinak) se s objektem pracuje.

4. Skládání – Objekt může obsahovat jiné objekty.

5. Delegování – Objekt může využívat služeb jiných objektů tak, že je požádá o provedení operace.

6. **Dědičnost** – objekty jsou organizovány stromovým způsobem, kdy objekty nějakého druhu mohou dědit z jiného druhu objektů, čímž přebírají jejich schopnosti, ke kterým pouze přidávají svoje vlastní rozšíření. Tato myšlenka se obvykle implementuje pomocí rozdělení objektů do tříd, přičemž každý objekt je instancí nějaké třídy. Každá třída pak může dědit od jiné třídy (v některých programovacích jazycích i z několika jiných tříd).

7. **Polymorfismus** – odkazovaný objekt se chová podle toho, jaké třídy je instancí. Pokud několik objektů poskytuje stejné rozhraní, pracuje se s nimi stejným způsobem, ale jejich konkrétní chování se liší podle implementace. U polymorfismu podmíněného dědičností to znamená, že na místo, kde je očekávána instance nějaké třídy, můžeme dosadit i instanci libovolné její podtřídy, neboť rozhraní třídy je podmnožinou rozhraní podtřídy. U polymorfismu nepodmíněného dědičností je dostačující, jestliže se rozhraní (nebo jejich požadované části) u různých tříd shodují, pak jsou vzájemně polymorfní.

Objektový přístup programování trochu jinak:

- Modelování problému jako systému spolupracujících tříd

- Třída modeluje jeden koncept
- Třídy umožní generování instancí, objektů příslušné třídy
- Jednotlivé objekty spolu spolupracují, „posílají si zprávy“,
- Třída je „vzorem“ pro strukturu a vlastnosti generovaných objektů
- Každý objekt je charakteristický specifickými hodnotami svých atributů a společnými vlastnostmi třídy

## 1.1 Třída

Třída je základem uživatelského programu v jazyku Java (nebo obecně v OOP). Třída = koncept. Každá třída je reprezentována svými prvky, objekty dané třídy. Každá třída je charakterizována svými vlastnostmi, svými funkčními možnostmi a svými parametry. Třída tedy popise nějaký “objekt” reálného problému a udržuje jeho parametry a poskytuje metody pro práci s ním. Třída v bodech:

- musí být deklarována hlavní funkce main (NOTE: bylo v přednášce, ale není to pravda, třídy bez třídy main se běžně používá, např. jako knihovna). Funkce main musí být statická, volá se ještě před vytvořením nějaké instance, slouží pro spuštění programu, testování.
- jsou deklarovány další (static) funkce (či procedury) třídy, případně i v jiných třídách: projekt, package
- mohou být deklarovány statické proměnné, které jsou použitelné jako nelokální proměnné ve funkcích dané třídy
- program probíhá spuštěním příkazů metody main

Třída je definována množinou identifikátorů, které mají třídou definovaný význam:
– data: proměnné, konstanty (členské proměnné, datové složky, atributy)
– metody: pracovně funkce a procedury

### 1.1.1 Třída = předpis vs. Objekt = instance třídy

Třída jako šablona pro generování konkrétních instancí třídy, tzv. objektů. Jednotlivé instance třídy (objekty) mají stejné metody, ale nacházejí se v různých stavech – Stav objektu je určen hodnotami instančních, členských proměnných. Schopnosti objektu jsou dány instančními metodami třídy. V jazyku Java lze objekty (instance tříd) vytvářet pouze dynamicky pomocí operátoru new a přistupovat k nim pomocí referenčních proměnných (podobně jako u pole). Třída bez vytvořené instance (objektu) může „pracovat“ pouze „staticky“ (mohou být použity jen její statické metody či proměnné – procedurální přístup). Statické proměnné = společné pro všechny instance třídy.

### 1.1.2 Konstruktor třídy

Konstruktory = speciální metody pro generování instancí tříd, konkrétních objektů.

- vytvoří objekt
- případně nastaví vlastnosti objektu
- jméno konstruktoru je totožné se jménem třídy (jediná metoda začínající velkým písmenem)
- volání pomocí operátoru new, např. malyObdelnik = new Obdelnik(2,5);
- neosahuje návratový typ - nic nevrací, vytváří objekt
- není-li konstruktor vytvořen, je vygenerován implicitní konstruktor s prázdným seznamem parametrů – je-li konstruktor deklarován, implicitní zaniká

#### 1.1.2.1 Přetěžování (nejen konstruktoru třídy)

Umožňuje objektům volání jedné metody se stejným jménem, ale s jinou implementací. Provádí se to tak, že se deklaruje více metod se stejným názvem, které se mohou lišit různým počtem, typem argumentů popř. jejich pořadím. Umožňuje volat vytvoření instance s různými parametry - různým počtem i typem parametrů. Př.:

```
public class Obdelnik {


    // vsechny konstruktory lze pouzit zvenci pro vytvoreni objektu
    // kontruktor pro volani se vsemi tremi vlastnostmi,
    Obdelnik(int s, int v, Color c){
        sirka = s;
        vyska = v;
        barva = c;
    }
    // kontruktor pouze s rozmery - vyuziva prvni konstruktor pomoci this
    Obdelnik(int s, int v){
        this(s,v,Color.BLACK);
    }
    // prazdny konstruktor
    Obdelnik() {
        this(0,0,Color.BLACK);
    }

}
```

Konstruktor dále:

- Jméno konstruktoru je totožné se jménem třídy
- Konstruktor nemá návratovou hodnotu (ani void)
- Předčasně lze ukončit činnost konstruktoru return
- Konstruktor má parametrovou část jako metoda - může mít libovolný počet a typ parametrů
- V těle konstruktoru použít operátor this, odkaz na příslušný konstruktor s týmž počtem, pořadím a typem parametrů - nepíše se jméno třídy
- Konstruktor je zpravidla vždy public (!) – //třída java.lang.Math jej má private – proč? Protože je designována jako utilitní třída, která poskytuje statické metody (a atributy) a nevyžaduje vytvoření instance

### 1.1.3 Třída jako datový typ

Příkladem třídy jako datového typu je třída Complex

- hodnotami typu jsou komplexní čísla tvořená dvojicemi čísel typu double (reálná a imaginární část)
- množinu operací tvoří obvyklé operace nad komplexními čísly (absolutní hodnota, sčítání, odčítání, násobení a dělení)

### 1.1.4 Statické vs. Instanční metody

Třída může definovat dva druhy metod:

- statické metody – metody třídy, procedury a funkce
- instanční metody – metody objektů

Metody obou druhů mohou mít parametry a mohou vracet výsledek nějakého typu. Statická metoda označuje operaci (dílčí algoritmus, řešení dílčího podproblému), jejíž vyvolání (provedení) obsahuje jméno třídy, jméno metody a seznam skutečných parametrů jméno_třídy.jméno_metody(seznam skutečných parametrů). Statickým metodám třídy odpovídají v jiných jazycích procedury (nevracejí žádnou hodnotu) a funkce (vracejí hodnotu nějakého typu). Instanční metoda označuje operaci nad objektem (instancí (!)) dané třídy, jejíž vyvolání obsahuje referenční proměnnou objektu, jméno metody a seznam skutečných parametrů referenční_proměnná.jméno_metody(seznam skut. parametrů).

- Statickým metodám třídy budeme i nadále říkat procedury a funkce
- Instančním metodám budeme zkráceně říkat metody

### 1.1.5 Operátor this

Každý objekt má implicitní operátor this, který obsahuje odkaz na "svou" instanci

- Hodnotou operátoru this je odkaz na objekt pro který byla metoda zavolána (implicitní parametr odkazu na objekt)
- Umožňuje přístup k vlastním instančním proměnným v instančních metodách
- Používá se v přetížených konstruktorech

Podobně funguje operátor this i pro metody:

- pokud je instanční metoda volána z jiné instanční metody té samé třídy, potom se volá pomocí operátoru this (operátor se může vynechat)
- pokud se volá metoda z jiného kontextu, uvádí se před jejím jménem přístup k příslušné instanci (tečka notace) např. předané parametrem
- nelze volat ze statické metody - u ní by nebylo jasné, kam this odkazuje

## 1.2 Objekt

Objekt - datový prvek, instance třídy, dynamicky vytvořen podle „vzoru“- třídy (viz sekce Třída). Objekt si pamatuje svůj stav (v podobě dat čili atributů) a zveřejněním některých svých operací (nazývaných metody) poskytuje rozhraní, jak s ním pracovat. Objekt je strukturován tzn. skládá se z jednotlivých položek tzv. atributů. Objekt = heterogenní objekt skládající se z položek různého typu (x pole, pole se skládá z položek stejného typu). Složky objektu:

- atributy objektu (datové složky, členské proměnné (member variables)) - proměnné objektu, definují typ a jména vlastností objektu
- metody

Hodnota objektu je strukturovaná, tzn. skládá se dílčích hodnot, které mohou být obecně různého typu (heterogenní datová struktura – na rozdíl od pole). Objekt je tedy abstrakcí paměťového místa skládajícího se z částí, ve kterých jsou uloženy dílčí hodnoty - nazývají se položkami objektu (složkami, atributy, instančními proměnnými, fields, attributes). Položky objektu jsou označeny jmény, která mohou (ale nemusí) být třídou zveřejněna, zásadně se nezveřejňují. Pro manipulaci se zavádějí settery, gettery.

## 1.3 Zapouzdření

Zapouzdření v objektech znamená, že k obsahu objektu se nedostane nikdo jiný, než sám vlastník. Navenek se objekt projeví jen svým rozhraním (operacemi, metodami) a komunikačním protokolem. (Př. private proměnné -> pro manipulaci metody: nejjednodušší např. settery, gettery). Důležité pojmy:

- **Skládání objektů:** udržování odkazů na jiné objekty (objekt obsahuje jiné objekty).
- **Delegování:** Využívání služeb jiných objektů.

Zapoudření = Daný stav objektu je přístupný nebo měnitelný pouze prostřednictvím rozhraní poskytovaného objektem. Důsledkem zapouzdření je autorizovaný přístup k datům, při kterém zajistíme, že s daty objektu nebude možné z vnějšku třídy manipulovat jinak než pomocí metod této třídy, které tvoří komunikační rozhraní třídy. Zapouzdření je zajištěno pomocí modifikátorů:

1. public - lze je volat odkudkoli
2. neurčený - lze volat ze stejného package
3. protected - lze je volat pouze z metod stejné či odovozené třídy
4. private - lze je volat pouze z metod téže třídy

## 1.4 Dědičnost

Dědičnost je mechanismus umožňující

- rozšiřovat datové položky tříd (také modifikovat)
- rozšiřovat či modifikovat metody tříd

Dědičnost umožní

- vytvářet hierarchie tříd
- „předávat“ datové položky a metody k rozšíření a úpravě
- specializovat, „upřesňovat“ třídy

Dvě základní výhody dědění

- Dědění má praktický význam v znuvupoužitelnosti programového kódu
- Dědičnost je základem polymorfismu

Java dědí pouze od jediného předka. Mnohonásobné dědění se řeší pomocí rozhraní. Nejvyšší třída je Object, všechny třídy dědí třídu Object (metody: equals (standardní implementace neporovnává objekty, ale reference), toString, hashCode (stejné objekty by měly generovat stejný hashCode), clone..)

### 1.4.1 Příklad: Obdélník je případ kvádru (ne naopak)

Obdélník je „kvádrem“ s nulovou hloubkou

- Potomek se deklaruje pomocí klauzule extends
- Obdelnik převezme proměnné sirka, vyska, hloubka metody hodnotaSirky, delkaUhlopricky
- Konstruktor se dědí, parametr hloubka se nastaví do nuly
- Objekty Obdelnik mohou využívat proměnné sirka, vyska a hloubka, metody hodnotaSirky a delkaUhlopricky

```
public class Kvadr2 {

    public int sirka;
    public int vyska;
    public int hloubka;
    public Kvadr2(int sirka, int vyska, int hloubka) {
        this.hloubka = hloubka;
        this.sirka = sirka;
        this.vyska = vyska;
    }
    public int hodnotaSirky() {
        return sirka;
    }
    public double delkaUhlopricky() {

        double pom = (sirka * sirka) + (vyska*vyska) + (hloubka * hloubka);
        return Math.sqrt(pom);
    }

}
class Obdelnik2 extends Kvadr2{

    public Obdelnik2(int sirka, int vyska) {
        super(sirka, vyska, 0);
    }

}
```

### 1.4.2 Hiearchie tříd

Třída Tpod, která je podtřídou třídy Tnad, dědí vlastnosti nadtřídy Tnad a rozšiřuje je o nové vlastnosti; některé zděděné vlastnosti mohou být v podtřídě modifikovány. Pro instanční metody to znamená:

1. každá metoda třídy Tnad je i metodou třídy Tpod, v podtřídě však může mít jinou implementaci (může být zastíněna - override = Podtřída obsahuje metodu se stejným názvem i stejnými parametry, metody nadtřídy zastíní vlastní implementací. Př. typicky toString())
2. v podtřídě mohou být definovány nové metody

Pro strukturu objektu to znamená:

- instance třídy Tpod mají všechny členy třídy Tnad a případně další

Pro referenční proměnné to znamená:

- proměnné typu Tnad může být přiřazena reference na objekt typu Tpod
- na objekt referencovaný proměnnou typu Tnad lze vyvolat pouze metodu deklarovanou ve třídě Tnad; jde-li však o objekt typu Tpod, metoda se provede tak, jak je dáno třídou Tpod
- hodnotu referenční proměnné typu Tnad lze přiřadit referenční proměnné typu Tpod pouze s použitím přetypování, které zkontroluje, zda referencovaný objekt je typu Tpod

Vztah nadtřída – podtřída je tranzitivní = jestliže je x nad třídou y a y je nad třídou z, pak je x nadtřídou z

## 1.5 Kompozice

Obsahuje-li deklarace třídy členské proměnné objektového typu, pak mluvíme o kompozici objektů. Kompozice vytváří hierarchii objektů, nikoli však dědičnost. Například třída, která obsahuje jako členskou proměnnou integer, ale hlavně i například Datum (další, jiný objekt). (Kompozice označována jako struktura HAS ("má", dědičnost jako ISE "je"). Kompozice objektů je tvořena atributy objektového typu, pouze je skládá

## 1.6 Polymorfismus

V programovacím jazyce se jedná o možnost volat stejné metody u různých objektů, aniž bychom věděli, jakého přesně jsou typu. Navíc může mít stejná metoda u různých objektů odlišný význam. To je možné díky tomu, že vždy známe společného předka těchto různých objektů. Tím může být třída, abstraktní třída nebo rozhraní.

```
// soubor Osoba.java
public class Osoba {

    public int fce () {
        return 10;
```

```
    }
}
// soubor Zamestnanec.java
public class Zamestnanec extends Osoba {

    public int fce () {
        return 20;
    }

}
// jiny soubor
Zamestnanec a = new Zamestnanec();
Osoba b = new Zamestnanec();
Osoba c = new Osoba();
System.out.println(a.fce());    // vypíše 20
System.out.println(b.fce());    // vypíše 20
System.out.println(c.fce());    // vypíše 10
```

## 1.7 Abstraktní třídy

V některých situacích je výhodné vytvořit jedinou bázovou třídu pro více tříd odpovídajících konkrétním objektům, i když tato samotná bázová třída žádnému konkrétnímu objektu neodpovídá. Může ovšem nést některá data a poskytovat metody, které jsou odvozeným třídám společné. Taková třída se pak nazývá abstraktní a je označena klíčovým slovem abstract. Překladač jazyka Java pak zajistí, že instanci abstraktní třídy nelze operátorem new přímo vytvořit, mohou se vytvářet pouze instance konkrétních tříd.

Abstraktní třída může deklarovat některé společné metody a poskytovat jejich základní implementaci. Pokud odvozená třída takovou metodu nepředefinuje, pak se pro její instance použije implementace poskytnutá v bázové třídě.

Mohou však nastat i situace, kdy skutečně vyžadujeme, aby odvozené třídy určitou metodu vždy definovaly. Takovou metodu pak také nazýváme abstraktní a označujeme klíčovým slovem abstract, navíc u ní není uvedeno tělo a hlavička metody je zakončena středníkem. Pokud odvozená třída některou abstraktní metodu neimplementuje, musí být také označena jako abstraktní. Tím je zajištěno, že instance konkrétních tříd mají všechny metody implementované.

```
abstract class Obrazec {

    public Obrazec(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public abstract double obvod();
```

```
    public abstract double obsah();
    protected double x;
    protected double y;

}
```

## 1.8 Rozhraní

Druhou velmi podobnou konstrukcí jsou rozhraní (klíčové slovo interface). Základním rozdílem je, že interface obsahuje pouze konstanty a metody bez těla (v tomto případě je neoznačujeme slovem abstract). Rozhraní je kontrakt, který specifikuje operace, které má třída splňovat, a který se již nezabývá tím, jak toho bude konkrétně dosaženo.

Velkou výhodou rozhraní oproti abstraktním třídám je, že každá třída může implementovat až mnoho rozhraní, avšak vždy maximálně jednu třídu.

Zatímto pro dědění tříd (a dědení interfaců mezi sebou) využíváme v hlavičce klíčové slovo extends, tak pro implementaci rozhraní používáme slovo implements.

```
public interface Visualni {

    public void vykresli (Graphics kam);

}
```

### 1.8.1 Rozhraní jako typ proměnné

V Javě lze definovat proměnnou typu reference na rozhraní, ve které může být uložena libovolná třída, která toto rozhraní implementuje. Jména rozhraní lze používat jako referenční datové typy stejným způsobem jako jména tříd.

```
Visualni visualniObjekt = new VisualniKruznice();
visualniObjekt = new VisualniCtverec();
```

## 1.9 Výčtové typy - enum

Výčtové typy jsou speciální třídy zavedené pro větší bezpečí a pohodlí, v nejjednodušší variantě se definují (příklad):

```
enum Den {SUN, MON, TUE, WED, THU, FRI, SAT;}
for ( Den d : Den.values( ) )

    System.out.println( d.ordinal( )+ " " +d.name( ) );
```