

Design time

- vytváří se program
- metadata
- vytváří se např. třída

Run time

- běh programu
- ukládá se do paměti počítače
- vytváří se např. objekt

rysy javy - zavedení jasného pořádku a etikety

- přísné typování jmen
- abstrakce (abstraction) = zobecnění pohledů a částečná řešení
- dědičnost (inheritance) = získávání vlastností a schopností děděním
- polymorfismus = jednotné zacházení s objekty různých tříd
- zapouzdření (encapsulation) = skrývání části tříd a interfejsů

základ javy - třída (class)

- interfejs (interface = rozhraní či interfejs)
- výčet (enum)

uloženo v **package**

java.lang = základní nutný balíček

- obsahuje třídy, bez kterých nemůže java existovat a fungovat

zdrojové soubory „*.java“ obsahují

- jeden příkaz package – je vždy první
- několik příkazů (static), import pro snadnější přístup k jiným balíčkům
- definice tříd, interfejsů a výčtů v libovolném pořadí
- komentáře dokumentační (před třídou a interfejsem) a ostatní (kdekoli)

třída

- úložiště statických položek tříd
- plán pro tvorbu objektů, definuje úplný popis objektu
- zdroj neprivatních členů pro dědění potomkům
- brána pro spuštění aplikace pomocí „public static void main...“
- syntaxe třídy:
*[public/protected...] [abstract] [static] class subJméno [extends/implements supJméno] { “tělo”
statické/nestatické atributy, inicializátory, metody; konstruktory, vnořené a vnitřní třídy/interfejsy, výčty
(enum), anotace }*
- tvorba tříd – zcela nová, děděním, použijí již vytvořený “vzor”

abstraktní třída

- má modifikátor abstrakt
- má alespoň jednu nebo všechny abstraktní metody
- metody nemají těla – místo těla jen „ ; “
- má alespoň jeden konstrukt, ale nelze vytvořit objekt
- je k vytváření plánů, vyjasňuje strukturu systému
- brána pro spuštění aplikace pomocí „public static void main...“

interface

- slouží jako norma či požadavek
- omezená abstraktní třída – má jen metody
- nemá konstruktory ani inicializátory
- nemůže být potomkem žádné třídy
- může být potomkem více interfejsů
- nemůže být finální
- syntaxe interfejsu:
*[public/protected...] [abstract] [static] class subTřída [extends/implements supTřída] { “tělo”
[public static final] = atributy inicializované v definici, [public abstrakt] = metody, ale ne static, vnořené a
vnitřní třídy/interfejsy, výčty (enum), anotace }*

	instanční prom.	tělo metody	abstrakt. metody	vícenásob. dědění
class	+	+	-	-
abstract class	+	+	+	-
interface	-	-	+	+

dědičnost (inheritance)

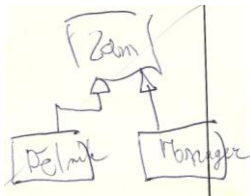
- usnadňuje přístup k existujícím třídám a interfejsům
- jedna třída je rozšířením druhé
- vytváření nových tříd a interfejsů s využitím vlastností a schopností již existujících
- extends (rozšiřuje) = třída – třída, interfejs – interfejs
- implements (splňuje) = třída – interfejs
- class TřídaB extends TřídaA, class TřídaA implements Interface
- nejvyšší sup = java.lang.Object
- každá třída sub má jeden super
- interfejs sub může mít více super = může být potomkem více interfejsů
- dědí se jen viditelné členy = public, protected – ne private a static
- nedědí se konstruktor, private a static členy/metody
- potomek může přidat nové členy a definovat vlastní konstruktory a inicializátory
- override = překrytí (přepis) zděděné metody novými metodami na podmínky subclassu, nelze překrýt statickou

překrytí x přetížení

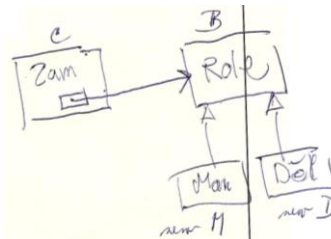
- překrytí = vyžaduje použití stejného názvu a stejného seznamu argumentů
- přetížení = musíme definovat metodu se stejným jménem, ale jiným seznamem argumentů

dědičnost (inheritance) x kompozice

- dědičnost = zejména přidávání method



- kompozice = zejména přidávání referenčních atributů
doporučuje se více



zastínění a překrytí

- lze v potomkovi u zděděného atributu
- u rodiče se překrytím metoda nezmění, je přístupná z potomka pomocí „super“
- statické atributy jsou přístupné přes jméno třídy, nelze zastínit a překrýt
- překrytí = overriding – pokud potomkovi nevyhovuje zděděná metoda, lze deklarovat metodu, která má shodnou signaturu, kovariantní návratový typ, nemění staticnost/nestaticnost, nezužuje modifikátor přístupu

polymorfismus

= mnohočlennost

- stejné zacházení s prvky různých tříd
- relace mezi referenčním typem a množinou referenčních typů

zapouzdření (encapsulation)

- míru zapouzdření určuje modifikátor viditelnosti členu či konstruktoru
- public = odevšad
- protected = ve vlastním balíčku a v potomcích, modifikátor se nepíše
- private = jen ve vlastní třídě
- k nedostupným členům lze z jiných tříd přistoupit JEN přes neprivátní metodu oné třídy
- v interfejsu lze zapouzdřit jen atributy jen úrovní „default“

getry a setry

- přístupová metoda k privátním atributům
- get (getter) – hodnotu vydávají, vracejí boolean
- set (setter) – hodnotu mění

statické atributy - inicializace

- připomínají metody bez hlavičky
- „mají“ jen tělo = static {...}
- mají přístup jen ke statickému kontextu
- nic nevracejí
- nelze je volat, nejsou členy třídy, nedědí se

konstruktor

- vytváří „na hladě“ objekt a dle parametrů sestavuje jeho atributy
- připomíná nestatickou nefinální metodu, která
 - má stejné jméno jako třída
 - nevyznačuje návratový typ, ani void
 - vrací JEN referenci svého typu na objekt, který vytvořil
 - má modifikátory viditelnosti
 - není členem třídy a nedědí se
 - konstruktory lze přetěžovat
- každá třída má alespoň jeden konstruktor
- pokud není, vytvoří se skrytý `public Třída() {super();}`

vzájemné volání konstruktorů – 3 případy

- nejprve existující konstruktor volá přímého předka
`public Třída() {super(...);...}`
- nejprve existující přetížený konstruktor volá konstruktor téže třídy
- lze použít i několikrát, ale nesmí dojít k cyklu
`public Třída() {this(...);...}`
- v ostatních případech se na začátek skrytě přidá volání `super()`;
`public Třída(...) {...}` nebo `public Třída(...) {super();...}` – ekvivalentní zápisy

objekt = instance třídy

- vytváří se v operační paměti na tzv. haldě
- obsahuje nestatické = instanční atributy primitivních nebo referenčních typů
- vytváří se konstruktorem a event. nestatickými inicializátory
- každý objekt je jedinečnou referencí popisující jeho třídu
- ruší je jen „garbage collector“

vytvoření objektu

- aktivací konstruktoru
`new Třída(...)` nebo `new Třída(...) {...}` – pole `new Pole[num]` nebo `new Pole[] {..., ..., ...}`
- klonováním pomocí metody `clone()`
- nestatickou metodou `java.lang.Class.newInstance()` – např.
`Class c = Class.forName(JménoTřídy);`
`Object o = c.newInstance();`
`JménoTřídy c = (JménoTřídy) x; //tzv. přetypování`

class Object

- jedna z nejdůležitějších tříd javy
- obsahuje 11 základních metod – `toString()`, `clone()`, `next()`, `equals()`, `hashCode()`, `finalize()`, `getClass()`, `notify()`, `notifyAll()`, `wait()`;...
- super class všech objektů
- všechny třídy automaticky dědí její vlastnosti
- metoda `String toString() {kód; return...;}`
např. `class Vector {`
 `String toString() {`
 `return x; }`
`}`

- v java.util
- add (Object o); = přidá do kolekce
- remove (Object o); = odebere
- boolean contains (Object o); = zda existuje, obsahuje
- int size (); = velikost
- iterator (); = prochází kolekcí
- atd. – má asi 20 metod
- má 4 základní rozšiřující interfejsy s těmito podtřídami

interface	metodou pole	spojový seznam	„hašování“	binární strom
List = pořadí, index	ArrayList	LinkedList		
Set = objekt se nemůže opakovat			HashSet	TreeSet
Deque = dvoukoncová fronta		LinkedList		
Que = fronta				

- *List* ~ *get* (*int index*) = vrát prvek na daném indexu
 - interface pro práci s kolekcí
- ```
iterator Collection extends Iterable {
 c.iterator (); }
```

```
interface Iterable {
 Iterator iterator ();
}
```

- Iterator je objekt, který umožní projít kolekci
- má dvě abstraktní metody `Next` a `hasNext`

```
interface Iterator {
 Object next (); //vezme prvek, vrátí a posune se dál
 boolean hasNext (); } //vrací boolean
```

- cyklus pro iterator

```
for (Iterator i = c.iterator(); i.hasNext();) {
 sout (i.next()); }
```

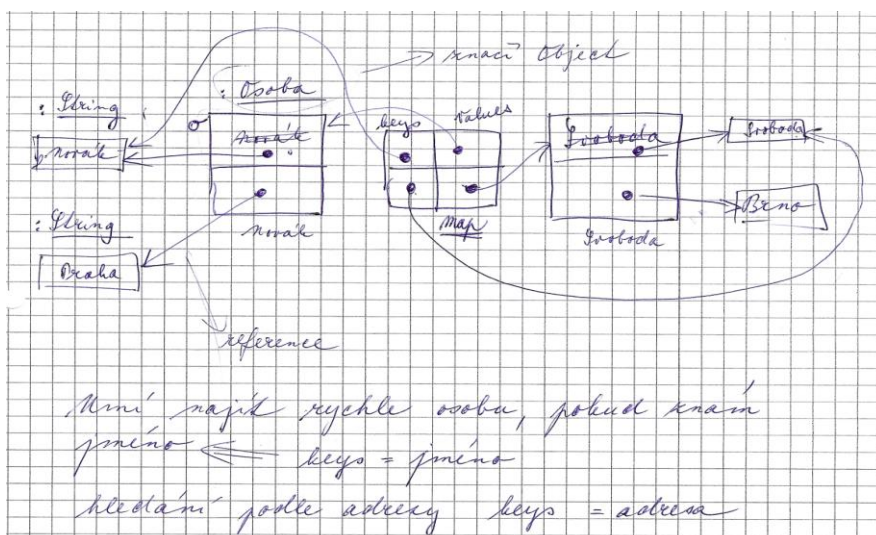
- Map = množina uspořádaných dvojic
- není v Collection
- má 2 základní metody
 

```
put (Object key, Object value); //místo add
get (Object key); //najde prvek,
 //velmi rychlá operace
```
- má 2 třídy
  - TreeMap* = operace má logaritmickou složitost, OMEZENÍ – klíče musí být komparovatelné
  - HashMap* =

```

Class Osoba {
String jmeno;
String adresa;
Map osoby – new HashMap();
Osoba o = new Osoba (“Novak”, “Praha”);
osoby.put (o.jmeno, o);
...
Osoba o = (“Svoboda”, “Brno”);}

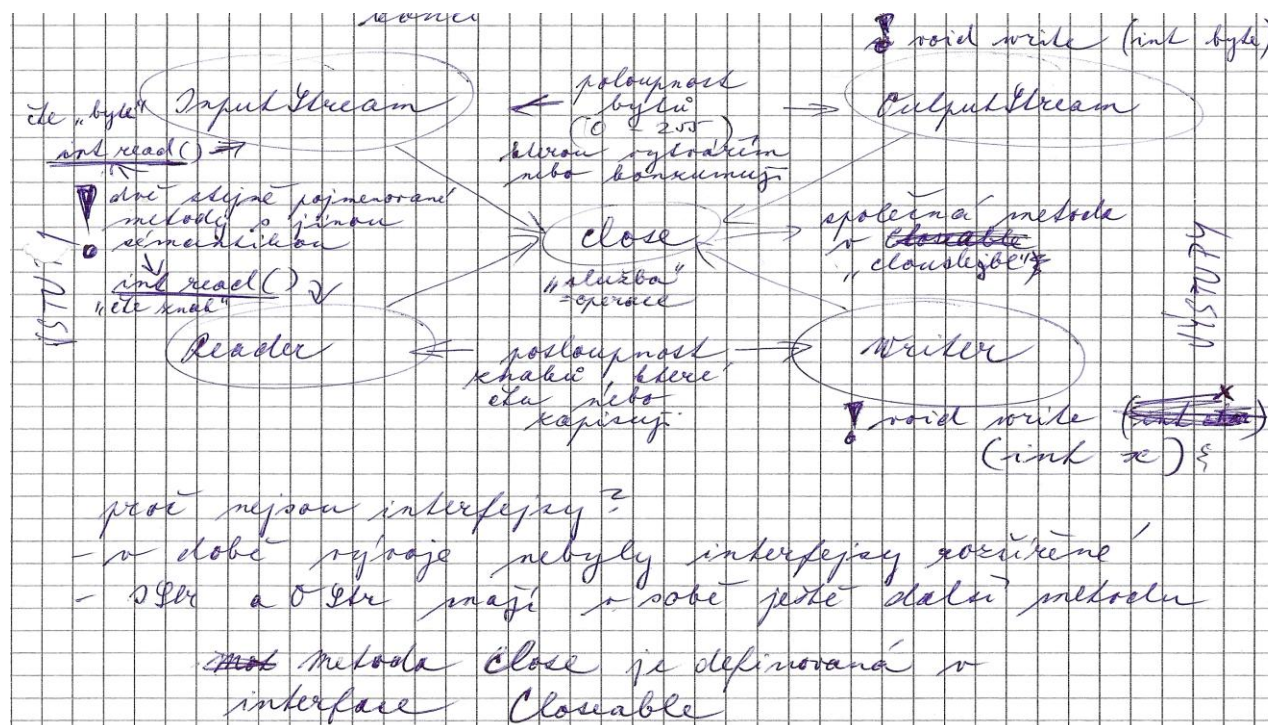
```



## java.io

= třída pro práci se streamy

- základní třídy = *Input Stream* x *OutputStream*, *Reader* x *Writer*



**Stream** = proud, fronta, ke které se přistupuje pouze z jedné strany

- jsou abstraktní = samy o sobě nejdou použít
- konec streamu = -1, null apod. – záleží na použité metodě

### Stream x Que

- *Stream* = data nejsou aktuálně k dispozici, jsou postupně generována
- *Que* = data jsou k dispozici v paměti

**Dekorování** = skládání = kompozice

- třída se nedědí, ale 2 třídy spolupracují
- „spolupracovník“ je uveden v konstruktoru,
- nejde o vztah sup – sub

### Př – kopírování streamů

```
static void copy (String in, String on){
 BufferedInputStream bis = new BufferedInputStream (new FileInputStream (in)); //vstup
 BufferedOutputStream bos = new BufferedOutputStream (new FileOutputStream (on)); //vystup
 for (int b = bis.read(); b >= 0; b = bis.read()) { //zakladni cyklus
 bos.write (b); //precne 1 bit, zapise 8 bitu !!!
 }
 bos.close ();
 bis.close ();
}
```

| vstup I = Input                                                     | výstup O = Output                         |                  |            |
|---------------------------------------------------------------------|-------------------------------------------|------------------|------------|
| FileInputStream                                                     | FileOutputStream                          | binární          | dekorátory |
| ByteArrayInputStream                                                | ByteArrayOutputStream                     |                  |            |
| DataInputStream                                                     | DataOutputStream                          |                  |            |
| ObjectInputStream                                                   | ObjectOutputStream                        | datové (textové) | dekorátory |
| BufferedReader<br>(čte celé řádky metodou <code>readLine()</code> ) | PrintWriter<br>(např. <code>sout</code> ) |                  |            |
| InputStreamReader<br>(umí kódovat znaky)                            | OutputStreamWriter                        |                  |            |
| FileReader                                                          | FileWriter                                |                  |            |
| StringReader                                                        | StringWriter                              |                  |            |

**Př** – máme HTML stránku kódovanou pro Win ve 1250 a chceme ji překódovat na UTF-8

```
static TrHtml (String in, String out) throws IOException {
 //vstupni kod
 BufferedReader ins = new BufferedReader (InputStreamReader (new FileInputStream (in)), "Windows-
1250");
 // vystupni kod
 PrintWriter pw = new PrintWriter (new OutputStreamWriter (new FileOutputStream (out)), "UTF-8");
 //metoda
 for (String line = ins.readLine(); line != null; line = ins.readLine()){
 line = line.replaceAll ("<", "<"); //priklad prikazu / nahrada jednoho znaku za jiny
 pw.println (line);
 } // for
 pw.close();
} // class
```

### **class RandomAccessFile**

- není Stream
- jedna ze základních operací seek (long pos) = nastavení na určitou pozici
- implementuje 2 interfejsy  
DataInputStream implements DataInput  
DataOutputStream implements DataOutput
- konec Random – hodí výjimku = Exception

### **class File**

- není Streams
- základní metody se souborem  
exists () = zda existuje  
size () = zjištění velikosti  
rename () = přejmenování  
delete () = smazání  
isFolder ()  
isFile ()  
listFile() = vrátí pole všech podadresářů nebo adresářů

**Př** = vypsání adresářů

```
static void dumpFiles (String rootPath){ //rootPath = cesta odkud chci vypisovat
 File root = new File (rootPath);
 pf (root, 1);
} // dump Files

// pomocna metoda na rekuzi
private static void pf (File f, int odsazeni) {
 String sps = "....."; //String mezer
 sout (sps.substring (0, odsazeni) + f);
 if (f.isDirectory ()) {
 File [] ch = f.listFiles ();
 for (File f:ch) {
 pf(f, odsazeni + 3);
 } // for
 } //if
} //void pf / konec rekurze
```

**java.net** = síťování

- IP = internet protokol = adresa počítače
- TCP/IP protokol
- socket = komunikace mezi 2 PC – každý konec má svůj OutputStream a InputStream (záleží na směru přenosu)
- komunikace mezi klientem (= aktivní část) a serverem (= pasivní část)

**Př**

```
package klient;
class Main {
public ... main () throw Exception {
//navazani spojeni se serverem
Socket s = new Socket ("..IP adresa...", 3333); // "localhost" = moje IP adresa
// 3333 = pr., nutno najit cislo volneho portu, nad 1000

// vstup
InputStream is = s.get InputStream (); // objekt s pro nas vytvori jiny objekt = metoda "factory"
// vystup
OutputStream os = s.get OutputStream ();

// dal pro priklad plati predpoklad, ze server i klient bezi na stejne platforme
BufferedReader br = new BufferedReade (new InputStreamReader (is));
PrintWriter pw = new PrintWriter (new OutputStreamWriter (os));
for (; ;){
String line = System.console ().readLine (); // precteni jedne radky
pw.println (line); // line predam na server
pw.flush ();
sout ("???" + br.readLine ());
} // for = nekonecna operace se serverem
} // main
} // class Main

package server;
class Main {
public ... main () {
ServerSocket ss = new ServerSocket (3333); // cislo portu musi byt shodne s klientem
Socekt s = ss.accept (); // ceka na prihlaseni klienta
.... dal uz je to shodne s klientem
for (; ;) {
String line = ... readLine ();
sout (line + "realy ???");
flash ();
} // for
} //
} // Main
```

## Výjimky = Exceptions

- mechanismus výjimek = mechanismus skoků
- = přenesení (řízení) běhu programu do jiného místa
- "mrtvý bod" = skočí do místa, kde není možno pokračovat, NUTNO OŠETŘIT ABY NENASTALO !!!
- jednoduché skoky = skoky uvnitř metod
- např. v cyklech if, switch nebo příkazy break, continue, return
- **throw** = "vržení výjimky" = skok

*throw <objekt výjimky>;*

### 2 požadavky na funkčnost

- bude to zavolaná metoda

*„catch klauzule“ = kde bude metoda pokračovat*

*catch (<parametr typu vyjimka>){ ... } // lze přirovnat k návěští*

- „try blok“ – příkaz throw musí být vykonán v rámci „try bloku“ // toto připomíná break

*výjimka může být vykonána v jakékoli metodě uvnitř*

*„try“ = výjimka se tzv. šíří z metody*

*try {*

*m ( m( ) { ... throw <...>;... } ); }*

### objekt výjimky x parametr výjimky

- objekt (reference) a parametr musí být kompatibilní

**Př**

*class ZeroVectorException extends Exception {*

*Zero Vector Exception (String name) {*

*super (name);*

*}*

*} // class*

*class Vector {*

*Vector (dbl x, dbl y) throws ZeroVectorException {*

*if (x == 0 && y == 0)*

*throw new ZeroVector Exception ("xxx");*

*}*

*}// class*

## Vlákna = Threads

- od vzniku vlákna probíhá více threadů najednou
- vše, co v javě běží, jsou thready
- thready MUSÍ vždy něco sdílet – sdílí společnou paměť
- všechny sdílí stejné objekty
- použití threadů

na straně serveru

- aby mohl paralelně obsluhovat více klientů najednou
- aby se daly použít vícejádrové procesory

na straně klienta

- **synchronizace** = “drží si pevně zámek” = jiný thread nemůže běžet, dokud synchronizace nedoběhne a “nepustí zámek”
- **join** = zpětné spojení jednotlivých threadů
- **metoda yield** = thread se dobrovolně vzdá běhu (procesoru) a zablokuje se
- class Thread = nejdůležitější třída
- metody
  - run
  - currentThread = vzpíše jméno aktuálního Threadu

## Př na “run”

```
class MyThread extends Thread {
 MyThread (String name) { super (name);}

 public void run () {
 for (int i = 0; i <= 9;){
 sout (i + “.” = getName());
 try { Thread.sleep (1000); } // thread lze uspat, jde o statickou metodu, pri “sleep” bezi jiny thread
 // POZOR, kdo zavola thread, ten se uspi
 // time v milisekundach
 } catch (InterruptedException) {0} // prazdny blok jde pouzit JEN v tomto pripade = u InterruptedException
 // misto prazdneho bloku jde pouzit pro ladeni napr. e.printStackTrace();
 } // for
 } // run
}

class Main {
 public static... () { // tady chci spoustet thready
 new MyThread (“Honza”).run();
 new MyThread (“Pepa”).run();
 }
}
```

## Př na “currentThread”

```
Thread.currentThread
sout (Thread.currentThread().getName()); //odpoved je Main
```