JAVA

Třídy

Definice třídy

- úplná definice

```
[public] [abstract] [final] class Jmeno
  [extends Predek]
  [impelements SeznamInterfacu] {
    ... // telo tridy
}
```

- **`public`** – veřejná třída
- **`abstract`** – nesmí být vytvářeny instance
- **`final`** – nelze vytvářet potomky

Konstruktor

- konstruktor
  - inicializace objektu
- deklarace
  - jmenuje se jako třída
  - žádný návratový typ
  - modifikátor – pouze viditelnost
  - více konstruktorů
    - různé parametry
    - určí se podle parametrů za `new`

```
class MyClass {
  int value;
  public MyClass() { value = 10; }
  public MyClass(int v) { value = v; }
}
```

Likvidace objektů

- garbage collector

- metoda `finalize()`
  - je na každé třídě
  - zavolá se před likvidací objektu
  - **není** to **destruktor** jako v jiných jazycích **!!!**
  - neví se, kdy se zavolá
  - není zaručeno jeho volání
    - objekty nemusejí být zlikvidovány přes garbage collector
      - např. při ukončení programu
  - volání `finalize` se neřetězí

Deprecated (od Java 9)

Inicializace atributů

- inicializace v konstruktoru
  nebo
- přímá inicializace

```
class MyClass {
  int a = 5;
  float b = 1.2;
  MyClass2 c = new MyClass2();
  int d = fn();
  int e = g(f);  // spatne!
  int f = 4;
  ...
}
```

Inicializace: `static`

- pouze jednou
- před prvním přístupem nebo před vytvořením první instance třídy
- přímá

```
static int a = 1;
```

- static inicializator

```
class MyClass {
  static int a;
  static {
    a = 10;
  }
  ...
}
```

Inicializace: "ne-static"

- podobné jako `static` inicializátor
- nezbytné pro inicializaci *anonymních vnitřních tříd*

```
class MyClass {
   int a;
   int b;
   {
      a = 5;
      b = 10;
   }
   ...
}
```

Třídy: dědičnost

- určení předka – **`extends`** *`JmenoPredka`*

- jednoduchá dědičnost
  - pouze jeden předek

- třída **`java.lang.Object`**
  - každá třída je jejím potomkem
    - přímo nebo nepřímo
  - jediná třída která nemá předka

- vícenásobná dědičnost přes **Interface**

Polymorfismus

- polymorfismus ~ dědičnost
- přetypování
  - automaticky – potomek na předka

```
class A { /*...*/ }
class B extends A { /*...*/ }

A a = new B();
Object o = a;

B b = (B) o;
```

Polymorfismus – konstruktor

- konstruktor předka
  - `super()`
- jiný konstruktor stejného objektu
  - `this()`
- volání jiných konstruktorů
  - jen jako první příkaz a pouze jednou
- vždy se nejdřív volá konstruktor předka
  - i když není explicitně uveden
  - výjimka – `this()`
- třída bez definice konstruktoru
  - standardní konstruktor
    - jen volá předka

java.lang.Object

```
Object clone()
boolean equals(Object obj)
void finalize()
Class<?> getClass()
int hashCode()
void notify()
void notifyAll()
String toString()
void wait()
void wait(long timeout)
void wait(long timeout, int nanos)
```

Třídy: viditelnost členů

- určuje se pro každý prvek

- atributy a metody
  - **`public`**
    - viditelné odkudkoliv (pokud je viditelná i třída)
  - **`protected`**
    - viditelné ve stejném balíku a v potomcích
  - **`private`**
    - viditelné jen ve třídě, kde jsou definovány
  - bez uvedení
    - viditelné ve stejném balíku

- platí v rámci jednoho modulu
  - od Java 9

Třídy: další modifikátory

- **`final`**
  - atribut
    - konstantní
    - musí mít inicializátor
    - po inicializaci nelze do něj nic přiřadit
  - metoda
    - nelze v potomcích předefinovat
- **`transient`**
  - atribut
  - není součástí persistentního stavu objektu
- **`volatile`**
  - atribut
  - přistupuje nesynchronizovaně několik vláken
  - nelze provádět optimalizace

Třídy: modifikátory metod

- **`abstract`**
  - není implementace metody
  - třída musí být také **`abstract`**
    - od třídy nelze vytvářet instance
  - tělo metody – středník
- **`synchronized`**
  - volající vlákno musí nejdřív získat zámek na volaném objektu (pokud to je **`static`** metoda, tak na třídě)
- **`native`**
  - nativní metoda
  - implementace přímo ve strojovém kódu konkrétní platformy (externí knihovna)
  - tělo metody – středník
- **`static`**
  - viz předchozí přednáška

Třídy: modifikátory metod

- není modifikátor **`virtual`**
- všechny metody jsou virtuální
  - statické metody ***nejsou*** virtuální

```
public class A {
  public void foo() {
    System.out.print("A");
  }
}

public class B extends A {
  public void foo() {
    System.out.print("B");
  }
}
....
A a = new B();
a.foo();     // vypíše B
```

```
public class As {
  public static void foo() {
    System.out.print("A");
  }
}

public class Bs extends As {
  public static void foo() {
    System.out.print("B");
  }
}
....
A a = new B();
a.foo();     // vypíše A
```

Statické metody

- statické metody se volají na třídě
  - nepatří žádnému objektu

```
class As {
  public static void foo() { ........ }
}

As.foo();
```

- lze je „volat“ i na objektu (instanci třídy);
  ve skutečnosti se vezme typ podle reference
  - hodnota objektu je nepodstatná
  - typ (a tedy která statická metoda bude volaná) se určí už v době překladu
    - viz předchozí slide

this

- reference na objekt právě volané metody
- lze použít pouze v těle metod nebo inicializátoru objektu

```
public class MyClass {
   private int a;
   public MyClass(int a) {
      this.a = a;
   }
}
```

super

- přístup k členům přímého předka
- pokud S je přímý předek C

  `((S) this).name` ~ `super.name`
- nelze použít `super.super`

```
class T1 { int x = 1; }
class T2 extends T1 { int x = 2; }
class T3 extends T2 {
   int x = 3;
   void test() {
      System.out.println(x);              // 3
      System.out.println(super.x);        // 2
      System.out.println(((T2)this).x);   // 2
      System.out.println(((T1)this).x);   // 1
   }
}
```

super

- `super` lze použít i na metody
- POZOR – nefunguje přetypování `this`
  - program lze přeložit, ale bude se rekurzivně volat stejná metoda

```
class TX1 {
  public void foo() { /*…*/ }
}
class TX2 extends TX1 {
  public void foo() { /*…*/ }
}
public class TX3 extends TX2 {
  public void foo() {
    ((TX1) this).foo();
    System.out.println("TX3.foo()");
  }
}
```

Java

Interface

Interface

- pouze definice rozhraní
- žádná implementace
  - od Java 8 i implementace
- může obsahovat
  - hlavičky metod
  - atributy
  - vnitřní interface

```
public interface Iterator {
   boolean hasNext();
   Object next();
   void remove();
}
```

Interface: atributy

- implicitně jsou `public`, `static` a `final`
- musejí být inicializovány
- v inicializaci nelze použít `super` a `this`

```
public interface Iface {
   int a = 5;
   String s = "ahoj";
}
```

Interface: metody

- bez implementace
  - implicitně jsou **`abstract`** a **`public`**
  - nelze definovat
    - **`synchronized`**
    - **`native`**
    - **`final`**
- **`default`** metody
  - od Java 8
  - obsahují implementaci
  - vhodné pro rozšiřování interfaců
- **`static`** metody
  - od Java 8
  - stejné jako static metody ve třídách

Interface: dědičnost

- vícenásobná dědičnost

```
interface Iface1 { ... }
interface Iface2 { ... }

interface Iface3 extends Iface1, Iface2
  { ... }
```

Třídy a interface

- třídy implementují interface

```
public interface Colorable {
  void setColor(int c);
  int getColor();
}
public class Point { int x,y; }
public class coloredPoint extends Point
                          implements Colorable {
  int color;
  public void setColor(int c) {
    color = c; }
  public int getColor() { return color;}
}

Colorable c = new ColoredPoint();
```

Třídy a interface

- třída musí implementovat všechny metody interface s výjimkou `default` metod
  - neplatí pro `abstract` třídy
- jedna metoda ve třídě může implementovat více interfaců

```
interface A { void log(String msg); }
interface B { void log(String msg); }

public class C implements A, B {
  public void log(String msg) {
    System.out.println(msg);
  }
}
```

Interfacy a default metody

- implementace ve třídě má vždy přednost
- při implementaci dvou interfaců se stejnou **`default`** metodou nutno implementovat metodu ve třídě
  - jinak třída nelze přeložit

```
interface If1 {
  default void foo() {…}
}
```

```
interface If2 {
  default void foo() {…}
}
```

```
class Mixed implements
                If1, If2 {
}
```

nelze přeložit

Interfacy a default metody

- nelze definovat default metody pro public metody z java.lang.Object

```
interface Iface {
  public default boolean equals(Object obj) {
    return false;
  }
}
```

- implementace ve třídě má vždy přednost
  - i zděděná

Interfacy a default metody

```
interface If1 {
    default void foo() {
        System.out.println("interface");
    }
}
```

```
class A {
    public void foo() {
        System.out.println("class");
    }
}
```

```
class B extends A implements If1 {
    public static void main(String[] args) {
        B b = new B();
        b.foo();   // -> "class"
    }
}
```

Java

Pole

Definice pole

- pole ~ objekt
- proměnná ~ reference

```
int[]      a;     // pole
short[][]  b;     // dvourozm. pole
Object[]   c,     // pole
           d;     // pole
long       e,     // skalar
           f[];   // pole
```

Inicializace pole

- "statická"

```
int[] a = { 1, 2, 3, 4, 5 };
char[] c = { 'a', 'h', 'o', 'j' };
String[] s = { "ahoj", "nazdar" };

int[][] d = { { 1, 2 }, { 3, 4 } };
```

Incializace pole

- dynamická

```
int[] array = new int [10];
float[][] matrix = new float[3][3];
```

- nemusí se zadávat všechny rozměry
  - stačí několik **prvních** rozměrů
  - místo ostatních prázdné závorky

```
float[][] matrix = new float[3][];
for (int i=0;i<3;i++)
  matrix[i] = new float [3];

// spatne
int[][][][] a = new int[3][][3][];
```

Inicializace pole

- "nepravoúhelníková" pole

```
int a[][] = { {1, 2}, {1, 2, 3}, {1, 2, 3,
  4, 5} };


int b[][] = new int [3][];
for (int i=0; i<3; i++)
  b[i] = new int [i+1];
```

Inicializace pole

- při vytváření pole se nevolá žádný konstruktor
- prvky ve vytvořeném poli (pomocí new) – defaultní hodnoty
  - reference – null
  - int – 0
  - ...
- výrazy při vytváření pole (**`new`**) – postupně plně vyhodnocovány zleva

```
int i = 4;
int ia[][] = new int[i][i=3];
// pole 4x3
```

Přístup k poli

- `pole[index]`
- indexy pole – vždy `0..délka-1`
- vždy se kontrolují meze
  - nelze vypnout
  - při překročení – výjimka `ArrayIndexOutOfBoundsException`
- délka pole – atribut length

```
int[] a = { 1, 2, 3 };
for (int i=0; i < a.length; i++) {
   ....
}
```

Pole ~ objekt

- `int[] intArray = new int [100];`
- `String[] strArray = new String [100];`

- pole je objekt

```
Object o1 = strArray;    // OK
Object o2 = intArray;    // OK
```

- ale

```
Object[] oa1 = strArray; // OK
Object[] oa2 = intArray; // špatně
```

Pole ~ objekt

```
Object[] oa = new Object [2];
oa[0] = new String("hello");
oa[1] = new String("world");

String[] sa1 = oa;  // špatně

String[] sa2 = (String[]) oa;
   // také špatně
   // lze přeložit, ale chyba při běhu
```