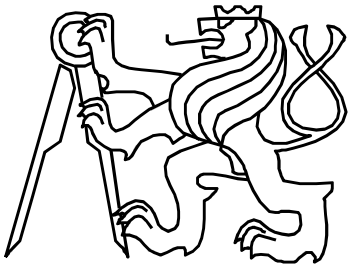
Enterprise Java (BI-EJA)
Technologie programování v jazyku Java (X36TJV)

Ing. Zdeněk Troníček, Ph.D.

Katedra softwarového inženýrství

Fakulta informačních technologií ČVUT v Praze



Letní semestr 2010/2011, přednáška č. 1
https://edux.fit.cvut.cz/courses/BI-EJA
https://edux.feld.cvut.cz/courses/X36TJV

Cíle předmětu

- Seznámit se s principy používanými při tvorbě podnikových aplikací
- Získat přehled o technologiích v Java Enterprise Edition 6
- Naučit se rozvrhnout si práci a dodržovat termíny

Program přednášek

14.2.	Generické typy, anotace, RMI
21.2.	JEE, servlety, JSP
28.2.	JSF
7.3.	JPA, EJB
14.3.	Transakce, security
21.3.	JMS, MDB, JWS
28.3.	JAX-WS, JAX-RS
4.4.	JavaFX
11.4.	JMX, monitorování
18.4.	Aplikační server, clustering
25.4.	-
2.5.	Spring Framework
9.5.	Performance

Hodnocení

- 12 testů na cvičení: 24 bodů
- Kontrolní body semestrální práce: 6 bodů
- Semestrální práce: 30 bodů
- Zkouška (min. 20)
  - Písemka: 20 bodů
  - Program: 20 bodů
- Hodnocení: 90 a více ... A
  80 – 89 ... B
  70 – 79 ... C
  60 – 69 ... D
  50 – 59 ... E

Generické typy (generics)

1.4:

```
List slova = new ArrayList();
```

5.0:

```
List<Integer> cisla = new ArrayList<Integer>();
```

```
List<String> slova = new ArrayList<String>();
slova.add( "prvni" );
slova.add( "druhy" );

String s = slova.get( 0 );
```

Terminologie

- java.util.List<E> - generický typ
- E - typová proměnná
- List<String> - parametrizovaný typ

Generický typ používá jednu nebo více typových proměnných.

java.util.List

```
public interface List<E> extends Collection<E> {

    boolean add( E o );

    E get( int index );

    List<E> subList( int fromIndex, int toIndex );

    ...

}
```

Type erasure (1)

```
List<String> slova = new ArrayList<String>();
slova.add( "java" );
String s = slova.get( 0 );
```

po překladu:

```
List slova = new ArrayList();
slova.add( "java" );
String s = (String) slova.get( 0 );
```

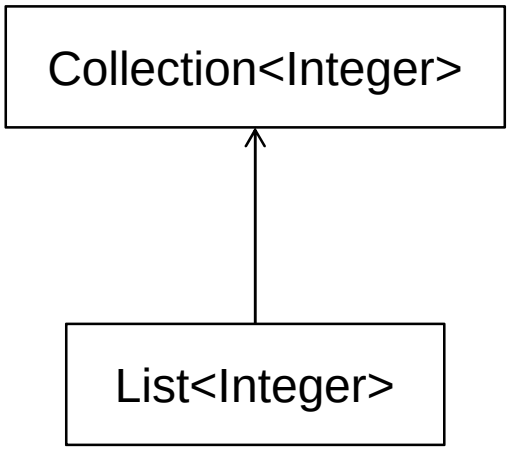
Type erasure (2)

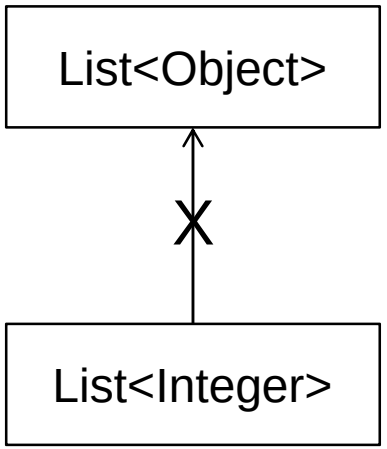
```
class Box<T> {
    T value;
    Box( T value ) { ... }
}
```

po překladu:

```
class Box {
    Object value;
    Box( Object value ) { ... }
}
```

Hierarchie typů



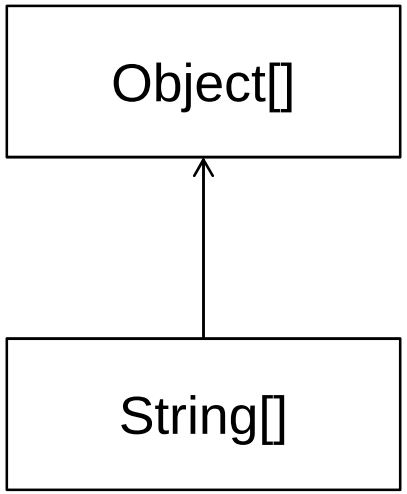



```
List<Integer> cisla1 = …;
Collection<Integer> cisla2 = cisla1;
```

```
// není možné toto:
List<Object> objekty = cisla1;
```

Pole v Javě

```
// typ String[] je potomkem Object[]
String[] jmena = new String[10];
Object[] objekty = jmena;

// ArrayStoreException:
objekty[0] = new Integer( 1 );
```



proměnná objekty:
- compile-time type: Object[] – používá překladač
- run-time type: String[] – používá JVM

Pole parametrizovaného typu

```
// hypotetický příklad:
List<Integer> cisla = new ArrayList<Integer>();
cisla.add( 1 );  // boxing

List<String>[] pole = new ArrayList<String>[10];

Object[] objekty = pole;
objekty[0] = cisla;
String s = pole[0].get( 0 );  // ClassCastException
```

Pole parametrizovaného typu není povoleno!

Wildcards (1)

metoda, která vytiskne prvky seznamu

Java 1.4:

```
static void printList( List s ) {
    for( int i=0; i < s.size(); i++ ) {
        Object p = s.get( i );
        System.out.println( i + ": " + p );
    }
}
```

Wildcards (2)

Java 1.5:

```
static void printList( List<Object> s ) {
    // stejné jako v 1.4
}
```

S jakým parametrem můžeme zavolat tuto metodu?

- List<Object>
- List<Integer>
- List<String>

Wildcards (3)

? = neznámý typ

List<?> = seznam něčeho

```
static void printList( List<?> s ) {
    // s.get() vrací Object
    Object o = s.get( 0 );
    // s.add() není povoleno
}
```

```
s.add( null );
```

Bounded wildcards (1)

metoda, která vrátí součet prvků v seznamu

```
static double sumList( List<?> s ) {
    double total = 0.0;
    for( Object o : s ) {
        Number n = (Number) o;
        total += n.doubleValue();
    }
    return total;
}
```

Bounded wildcards (2)

? extends Number = typ Number nebo libovolný potomek

```
static double sumList( List<? extends Number> s ) {
    // s.get() vrací Number
    // s.add() není povoleno
}
```

Type erasure (ještě jednou)

```
class Box<T extends Number> {
    T value;
    Box( T value ) { ... }
}
```

po překladu:

```
class Box {
    Number value;
    Box( Number value ) { ... }
}
```

Příklad

n-ární strom:

```
public class Tree<V> {
    private V value;
    private List<Tree<V>> p = new ArrayList<Tree<V>>();
    public Tree( V value ) { this.value = value; }
    public V getValue() { return value; }
    public void setValue( V value ) { this.value = value; }
    public void addChild( Tree<V> child ) { p.add( child ); }
}
```

Omezení typového parametru shora

```
public class Tree<V extends Comparable<V>> {
    private V value;
    public void join( Tree<V> t ) {
        int i = value.compareTo( t.value );
        ...
    }
}
```

Generická metoda

```
public class Util {
    public static <T> T[] fill( T[] p, T v ) {
        for( int i = 0; i < p.length; i++ ) {
            p[i] = v;
        }
        return p;
    }
}
```

Volání generické metody (1)

Překladač dokáže hodnotu typového parametru odvodit (tento proces se jmenuje *type inference*)

```
Boolean[] bools = Util.fill( new Boolean[10], Boolean.TRUE );

Object o = Util.fill( new Number[5], Integer.valueOf( 42 ) );
```

Volání generické metody (2)

```
Set<String> empty = Collections.<String>emptySet();

Set<String> empty = Collections.emptySet();

print( Collections.<String>emptySet() );
```

Omezení typového parametru zdola

? super T = typ T nebo libovolný předek

Collections.addAll():

```
public static <T> boolean addAll (
    Collection<? super T> c, T... elements ) { … }
```

Do jaké kolekce lze přidat hodnotu typu T?

Anotace

- Metadata = informace o informacích
- Standardní anotace (v balíku java.lang): Override, Deprecated, SuppressWarnings

```
@Override
public String toString() { ... }

@Deprecated
public class OldList { ... }

@SuppressWarnings( "unchecked" )
public void nonGenericsMethod() { ... }
```

Target & Retention policy

Target:

- ANNOTATION_TYPE
- CONSTRUCTOR
- FIELD
- LOCAL_VARIABLE
- METHOD
- PACKAGE
- PARAMETER
- TYPE

Retention policy:

- SOURCE – jsou odstraněny překladačem
- CLASS – jsou vloženy do souboru .class, ale JVM je nemusí načíst
- RUNTIME – musí být načteny za běhu, takže jsou přístupné přes reflection

Deklarace anotace

Deklarace:

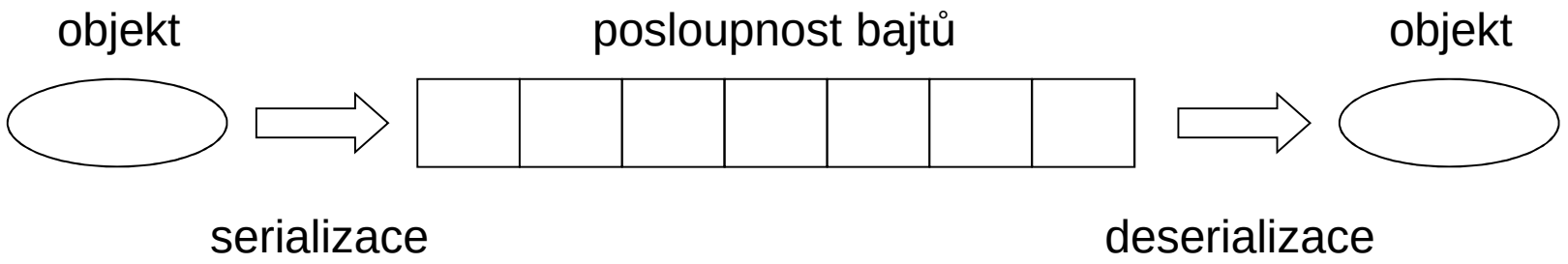
```
@Retention( RetentionPolicy.CLASS )
@Target( ElementType.METHOD )
public @interface Approved {
    public String msg() default "";
}
```

Použití:

```
@Approved( msg = "v1.2" )
public void print() { … }
```

Serializace

převod objektu na posloupnost bajtů



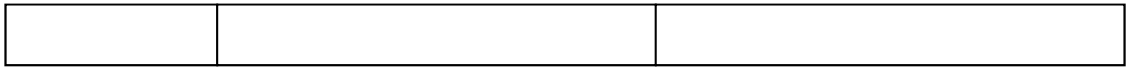
ObjectOutputStream & ObjectInputStream

```
ObjectOutputStream oos = new ObjectOutputStream(
    new FileOutputStream( "x.out" ));
oos.writeObject( new Date() );
oos.close();
```

```
ObjectInputStream ois = new ObjectInputStream(
    new FileInputStream( "x.out" ));
Date d = (Date) ois.readObject();
ois.close();
```

java.io.Serializable

```
public class Point implements Serializable {
    private int x;
    private int y;
    …
}
```



hlavička deskriptor třídy hodnoty atributů

(součástí není byte-kód)

Klíčové slovo transient

```
public class Point implements Serializable {
    private int x;
    private int y;
    private transient Thread t;
    …
}
```

Atribut serialPersistenceFields

```
public class Point implements Serializable {
    private int x;
    private int y;
    private Thread t;
    private static final ObjectStreamField[] serialPersistentFields = {
        new ObjectStreamField( "x", int.class ),
        new ObjectStreamField( "y", int.class )
    };
}
```

Metody writeObject a readObject

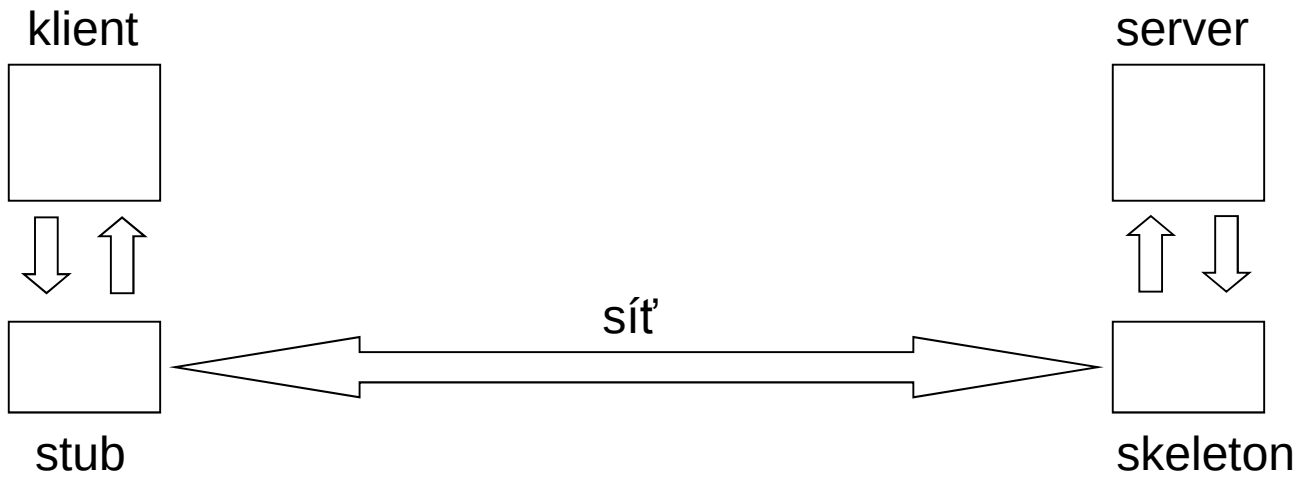
```
public class Point implements Serializable {
    …
    private void writeObject( ObjectOutputStream oos ) throws
        IOException { ... }
    private void readObject( ObjectInputStream ois ) throws
        IOException, ClassNotFoundException { ... }
}
```

Verze třídy (serialVersionUID)

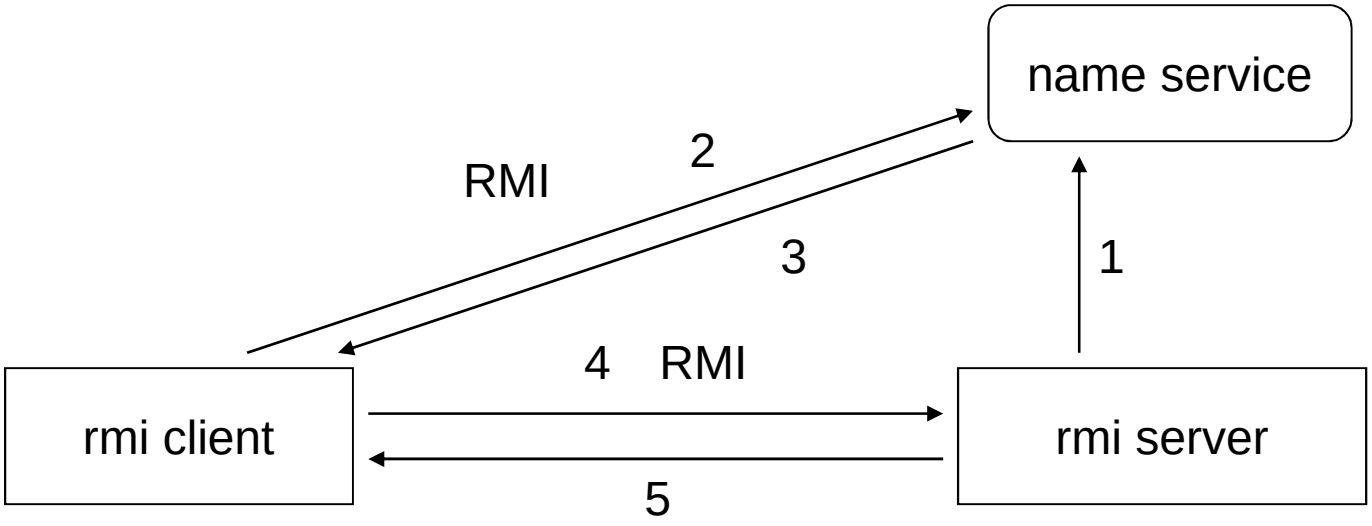
```
public class Point implements Serializable {

    private static final long serialVersionUID = 1L;

    …

}
```

Remote method invocation (RMI)

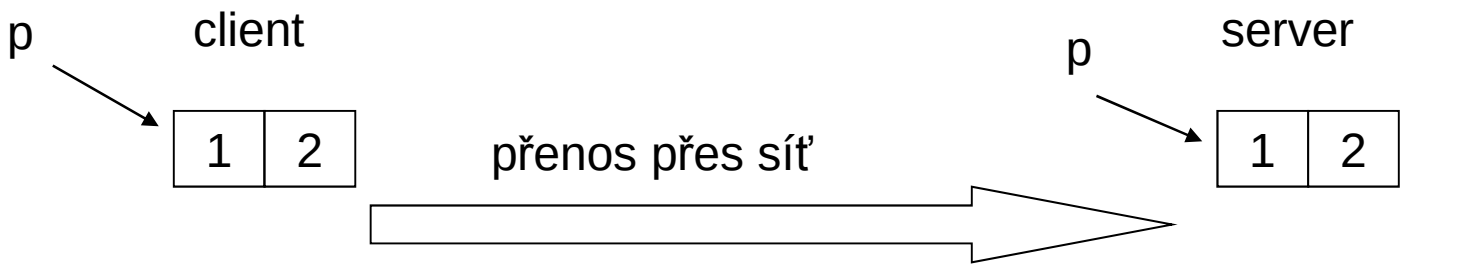
- umožňuje zavolat metodu na objektu v jiné JVM
- pro přenos parametrů a návratové hodnoty se používá serializace



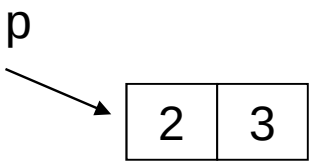
Vzdálené volání



Předávání parametrů



p.move(1, 1);



změna objektu p není automaticky viditelná na klientovi

Java Beans

Java Bean = znovupoužitelná softwarová komponenta, se kterou lze vizuálně manipulovat ve vývojovém nástroji

property = vlastnost beany; ovlivňuje vzhled nebo chování

Př.: property name typu String

- čtení: public String getName()
- změna: public void setName(String value)

Accessor & mutator

getter = metoda getX(), příp. isX()
setter = metoda setX(…)

property může být

- pro čtení (má *getter*)
- pro zápis (má *setter*)
- pro čtení i zápis (má *getter* i *setter*)

Otázky & odpovědi

Znáte NetBeans API a chcete pracovat na zajímavém projektu? Napište mi!