JavaBeans

Komponenty – přehled

- komponenta
  - znovupoužitelný kus kódu
  - charakterizována službami, které poskytuje a ***požaduje***
  - není přesná definice
- komponentové modely
  - JavaBeans
  - Enterprise JavaBeans (EJB)
  - ...
  - mnoho dalších komponentových modelů

JavaBeans – přehled

- JavaBeans poskytují
  - vlastnosti (properties)
  - události (events)
  - metody (methods)
- informace o komponentě
  - implicitní (introspekce)
  - explicitní
- propojení komponent
  - přes události
- persistence
  - implementace java.io.Serializable
- balíčky
  - do JAR

JavaBeans

- http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html
- specifikace
  - 1.00 1996
  - 1.01 1997
- jednoduchý komponentový model
  - Java objekty jako komponenty
  - jednoduchá manipulace a propojování v GUI vývojových prostředích
- definice
  - ***Java Bean is a reusable software component that can be manipulated visually in a builder tool***

JavaBeans

- jeden z cílů – jednoduchost
- model založený na **jmenných konvencích**
- *property*
  - jméno
    - př. foreground
  - metody pro přístup – set a get
    - void setForeground(Color c)
    - Color getForeground()
- *metody*
  - normalní metody
    - implicitně všechny public
- *events*
  - komunikace mezi komponentami
    - jedna komponenta "poslouchá" na události jiné komponenty

JavaBeans

- běh v různém prostředí
  - desing time vs. run time
- security
  - vše jako normální objekty
- typicky komponenta má GUI reprezentaci
  - můžou být i "neviditelné" komponenty bez GUI reprezentace
  - viditelné komponenty dědí od java.awt.Component
- žádná synchronizace
  - v případě potřeby si ji komponenta musí zajistit sama
- různé "pohledy" (views) na komponentu
  - komponenty složené z více tříd
    - není (a asi nebude) implementováno
  - `Component c = Beans.getInstanceOf(x, Component)`
  - nemělo by se používat normální přetypování

Události

- událost (event) – objekt
  - zdroj události
  - naslouchající objekt – listener
- různé události podle podle typu – různé objekty
  - předek java.util.EventObject
- listener
  - metoda, která se zavolá při výskytu události
  - interface java.util.EventListener
  - jeden listener muže mít více metod

Události - přehled

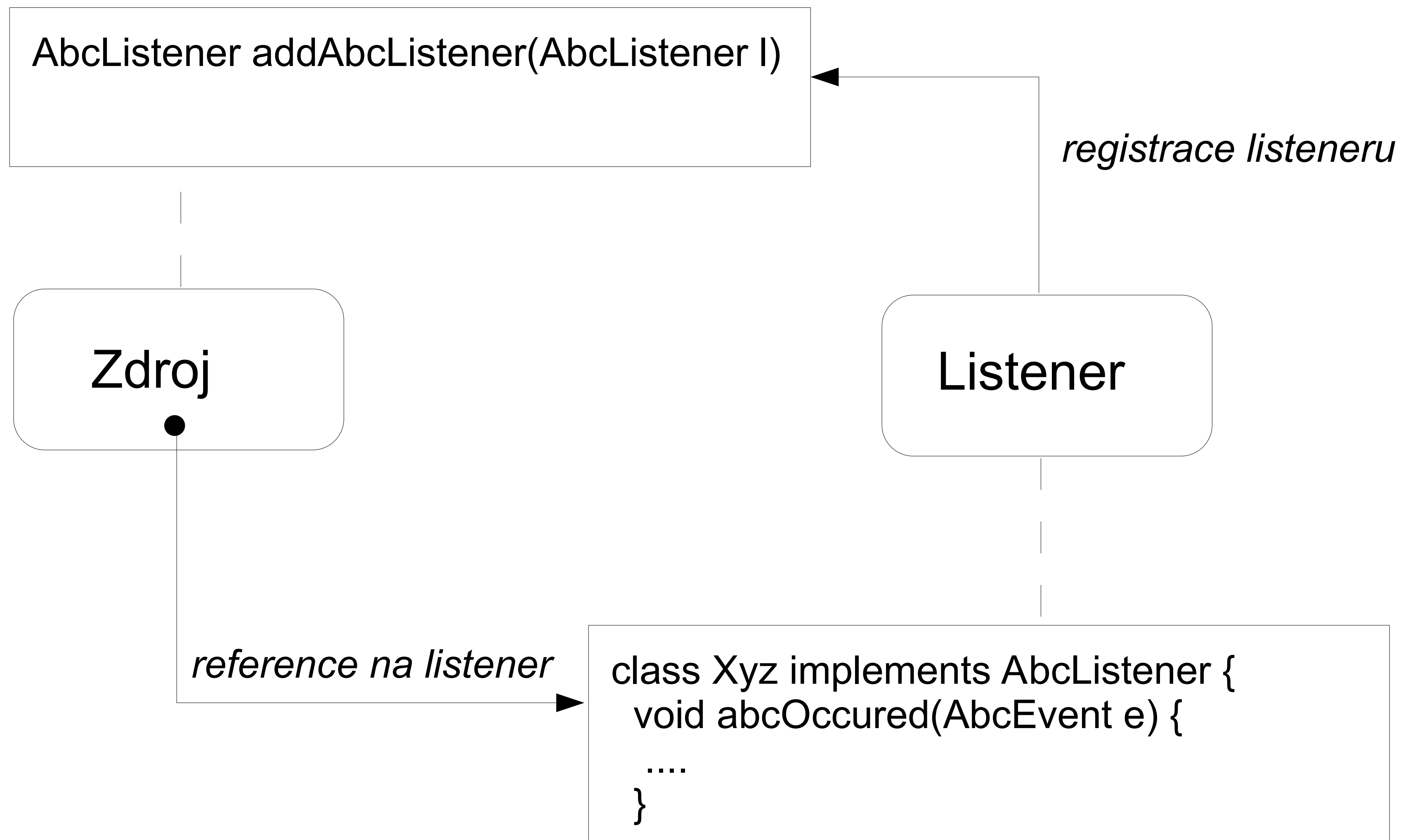
```
AbcListener addAbcListener(AbcListener l)
```

Zdroj

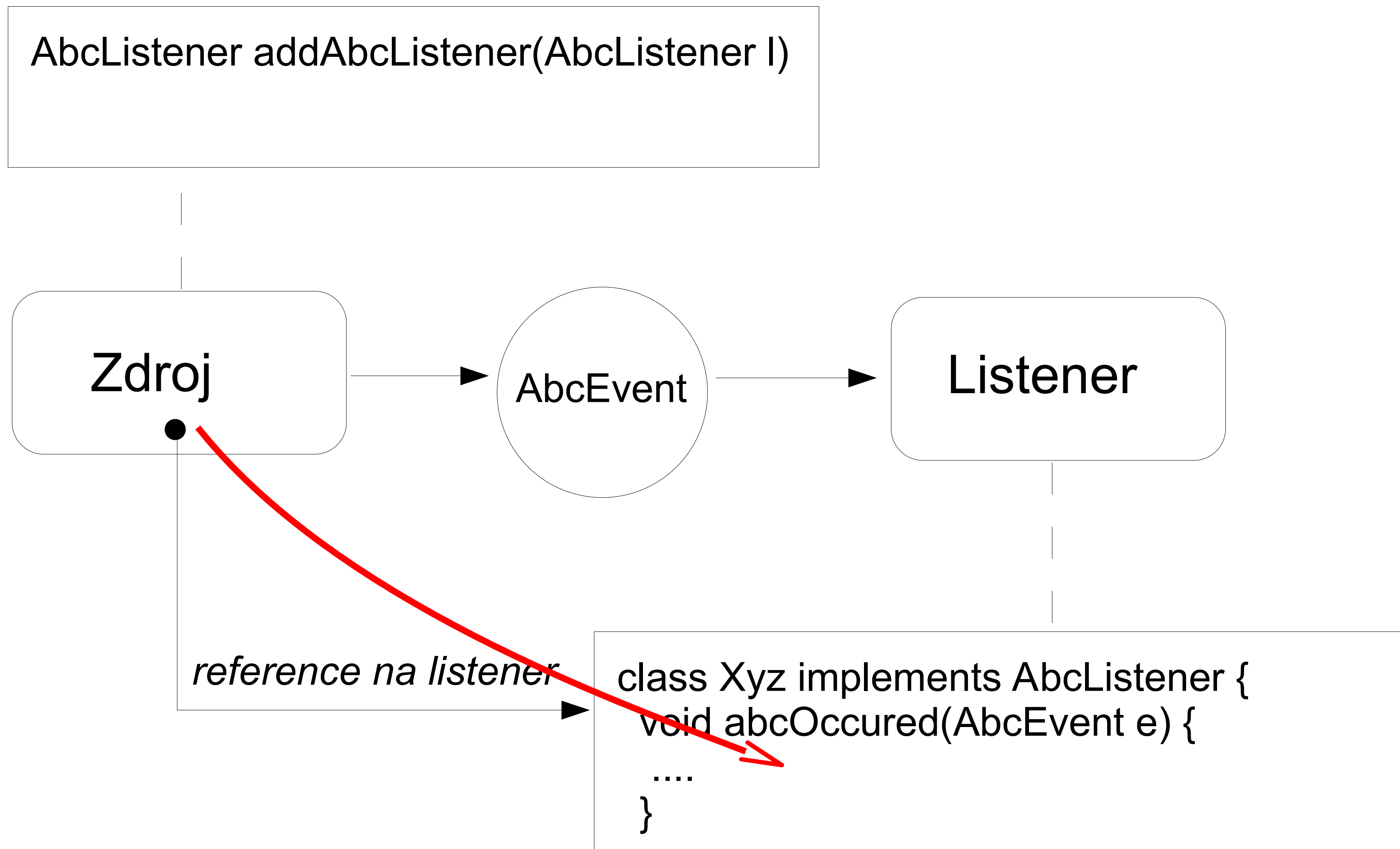
Listener

```
class Xyz implements AbcListener {
  void abcOccured(AbcEvent e) {
   ....
  }
```

Události - přehled



Události - přehled



Event objekt

- potomek java.util.EventObject
- typicky neměnitelný
  - privátní atributy
  - *get* metody

```
public class MouseMovedEvent extends EventObject {
  protected int x,y;

  public MouseMovedEvent(Component source, Point location) {
    super(source);
    x = location.x;
    y = location.y;
  }

  public Point getLocation() {
    return new Point(x, y);
  }
}
```

Listener

- interface – název končí na Listener (konvence)
  - dědí od java.util.EventListener
- definuje metody na obsluhu události
  - obecný vzor pro metodu
    - void *jakýEventNastal*(*EventObject* e)
- naslouchací objekt implementuje listener

```
public class MouseMovedListener implements EventListener {
  void mouseMoved(MouseMovedEvent e);
}
```

- jeden listener může definovat více metod pro související události
  - př. mouseMoved, mouseEntered, mouseExited
- metody můžou deklarovat výjimky
- parametr metody – událost
  - výjimečně seznam různých parametrů

Registrace listeneru

- komponenta, která může způsobovat události definuje metody pro registraci listenerů
  - odděleně pro každý typ
- obecný vzor
  - void add<TypeListeneru>(<TypListeneru> l)
  - void remove<TypeListeneru>(<TypListeneru> l)

```
public class Xyz {
  private ArrayList lst = new ArrayList();

  public void addMouseMovedListener(MouseMovedListener l) {
    lst.add(l);
  }
  public void removeMouseMovedListener(MouseMovedListener l) {
    lst.remove(l);
  }
  protected void fireMouseMovedEvent(int x, int y) {
    MouseMovedEvent e = new MouseMovedEvent(this, new Point(x,y);
    for (int i=0; i<lst.length; i++) {
      ((MouseMovedListener)lst.get(i)).mouseMoved(e);
    }
  }
}
```

Registrace listeneru

- unicast listener
  - nejvýše jeden zaregistrovaný listener
  - obecný vzor
    - void add<TypeListeneru>(<TypListeneru> l) ***throws TooManyListnersException***
    - void remove<TypeListeneru>(<TypListeneru> l)
- přidání/odebrání listeneru během obsluhy události
  - komu se event doručí?
    - záleží na implementaci
    - př. add*Listener* a remove*Listener* udělat synchronized a

```
protected void fireMouseMovedEvent(int x, int y) {
  Vector l;
  MouseMovedEvent e = new MouseMovedEvent(this,
                                          new Point(x,y);
  synchronized (this) { l = (Vector) listenres.clone(lst); }
  for (int i=0; i<l.length; i++) {
    ((MouseMovedListener)l.get(i)).mouseMoved(e);
  }
}
```

Event adaptor

- naslouchací objekt sám neimplementuje listener
  - vytvoří další objekt – adapter – který implementuje listener
  - zaregistruje adapter
  - adapter při obsluze události volá *nějaké* metody na naslouchacím objektu
- použití
  - filtrování události
  - reakce na různé události stejného typu
  - ....

Event adaptor

- příklad – Dialog
  - obsahuje 2 talčítka – OK a Cancel – obě generují událost ActionEvent
  - Dialog má metody
    - void doOKAction()
    - void doCancelAction()
  - dva adaptery – implementují ActionListener
    - OKButtonAdaptor
      - zaregistrovaný u OK tlačítka
      - metoda volá doOKAction na Dialogu
    - CancelButtonAdaptor
      - zaregistrovaný u Cancel tlačítka
      - metoda volá doCancelAction na Dialogu
- adaptory často jako (anonymní) vnitřní třídy

Vlastnosti (properties)

- vlastnost
  - jméno a typ
  - metody pro přístup
    - void setProperty(PropertyType c)
    - PropertyType getProperty()
- typ může být libovolný
  - výjimka u boolean property
    - místo *get* se používá *is*
    - př: void setEnabled(boolean b)
boolean isEnabled()
- metody můžou deklarovat výjimky

Indexed properties

- více-hodnotové vlastnosti (pole)
  - void setIndexedProperyt(int i, PropertyType c)
  - PropertyType getIndexedProperty(int i)
  - void setIndexedProperyt(PropertyType[] c)
  - PropertyType[] getIndexedProperty()

Bounded properties

- změna hodnoty vlastnosti způsobí událost
- událost PropertyChange
- listener PropertyChangeListener
- komponenta způsobí událost až ***po*** změně hodnoty vlastnosti
- pomocná třída PropertyChangeSupport
  - správa listenerů

Constrained properties

- jiná komponenta může zamítnout změnu hodnoty dané vlastnosti
- set metoda deklaruje PropertyVetoException výjimku
- při změně hodnoty komponenta způsobí událost VetoableChange
  - listener VetoableListener
  - pokud nějaký ze zaregistrovaných listenerů při obsluze události vyhodí PropertyVetoException, změna hodnoty se neprovede
- komponenta způsobí událost ***před*** změnou hodnoty vlastnosti
- pomocná třída VetoableChangeSupport

Bounded & Constrained props.

- lze, aby vlastnost byla jak *bounded* tak i *contained* naráz
  - pořadí zpracování
    1. VetoableChange událost
    2. pokud byla výjimka -> konec
    3. změna hodnoty
    4. PropertyChange událost
- při změně hodnoty na stejnou – nezpůsobovat žádné události
  - kvůli výkonu

Introspekce

- získávání informací o komponentě
  - vlastnosti
  - metody
  - události
- implicitní
  - podle vzorů introspekcí (java.lang.reflect)
  - vlastnosti
    - get a set metody
  - metody
    - všechny public
  - události
    - podle metod add*Listener* a remove*Listener*

Introspekce

- explicitní – *BeanInfo* třída
  - implementuje java.beans.BeanInfo interface
  - jmenuje se ***JmenoKomponenty*BeanInfo**

```
public interface BeanInfo {
    BeanDescriptor getBeanDescriptor();
    EventSetDescriptor[] getEventSetDescriptors();
    int getDefaultEventIndex();
    PropertyDescriptor[] getPropertyDescriptors();
    int getDefaultPropertyIndex();
    MethodDescriptor[] getMethodDescriptors();
    BeanInfo[] getAdditionalBeanInfo();
    java.awt.Image getIcon(int iconKind);
}
```

- typicky se BeanInfo třída vytváří jako potomek třídy SimpleBeanInfo
  - předpřipravená implementace

Introspekce

- BeanInfo nemusí popisovat všechny vlastnosti/události/metody
  - informace o ostatních lze získat přes introspekci
- při použití BeanInfo třídy se nemusí dodržovat konvence pro pojmenovávání
  - ale je to silně doporučeno

Introspector

- java.beans.Introspector
  - třída
  - standardní způsob pro získávání informací o komponentách
    - analyzuje BeanInfo (pokud existuje) i přímo třídu komponenty
    - analyzuje i předky komponenty

```
class Introspector {
  static BeanInfo getBeanInfo(Class<?> beanClass)
  static BeanInfo getBeanInfo(Class<?> beanClass,
                                          Class<?> stopClass)
  static String[] getBeanInfoSearchPath()
  static void setBeanInfoSearchPath(String[] path)
  ...
}
```

Property editor

- třída pro GUI editaci hodnot daného typu
  - v GUI vývojovém prostředí
- PropertyEditorManager
  - správce editorů
  - předregistrované editory pro základní typy
  - postup vyhledávání editoru pro daný typ
    1. hledání v explicitně zaregistrovaných
    2. třída, která se jmenuje stejně jako daný typ plus přípona Editor
    3. hledání v balících pro editory (lze nastavit přes PropertyEditorManager) – třída se stejným názvem jako v 2.
- property editor lze i zaregistrovat pro konkretní property v BeanInfo třídě

Customizer

- komponenta v GUI vývojovém prostředí
  - nastavování hodnot v tabulce vlastností
- pokud nelze vše nastavit přes vlastnosti => komponenta může mít Customizer
  - Dialog pro nastavení *nějakých* hodnot
  - měl by implementovat interface java.beans.Customizer a dědit od java.awt.Component
  - zaregistrován v BeanInfo

Persistence

- přes normální serializaci
- serializace
  - zcela normálně
- de-serializace
  - ClassLoader cl = this.getClass().getClassLoader();
  - MyBean b = (MyBean) Beans.instantiate(cl, "myPackage.MyBean");
  - nejdříve se hledá soubor se serializovanou komponentou
    - myPackage/MyBean.ser
  - pokud se nenajde, pak se přímo vytvoří instance komponenty

Balení komponent

- normální JAR soubor
- Manifest
  - speciální položky u popisu obsahu JAR souboru
  - Java-Bean: True
  - Depends-On: seznam tříd/souborů z JAR souboru
  - Design-Time-Only: True
- JAR typicky může obsahovat třídu komponenty i její serializovaný tvar (jmenoKomponenty.ser)

Java FX Beans
(pro srovnání)

Vlastnosti komponent (properties)

- interface Property<T>
  - void addListener(InvalidationListener listener)
  - void addListener(ChangeListener<? super T> listener)

  - void bind(ObservableValue<? extends T> observable)
  - void bindBidirectional(Property<T> other)
  - …

- implementace
  - class ObjectProperty<T>
  - class IntegerProperty
  - class BooleanProperty
  - class StringProperty
  - ...

Vlastnosti – příklad implementace

```
private StringProperty text =
                  new SimpleStringProperty("");

public final StringProperty textProperty() {
  return text;
}

public final void setText(String newValue){
  text.set(newValue);
}

public final String getText() {
  return text.get();
}
```

Vlastnosti – listenery

- `InvalidationListener`
  - volá se pokud současná hodnota vlastnosti přestala platit
  - umožňuje „líné“ vyhodnocení

```
void invalidated(Observable observable)
```

- `ChangeListener`
  - volá se při změně hodnoty vlastnosti
  - je potřeba spočítat i novou hodnotu
  - tj. neumožňuje „líné“ vyhodnocení

```
void changed(ObservableValue<? extends T>
             observable, T oldValue, T newValue)
```

Vlastnosti – propojování

- „binding“
- automatické aktualizování vlastnosti při změně jiné vlastnosti
  - interně implementováno pomocí listenerů

```
text1.textProperty().bind(text2.textProperty());

text1.textProperty().bindBidirectional(
                                    text2.textProperty());
```

- třída Bindings
  - statické metody pro snadné vytváření propojení

Práce s XML

Přehled

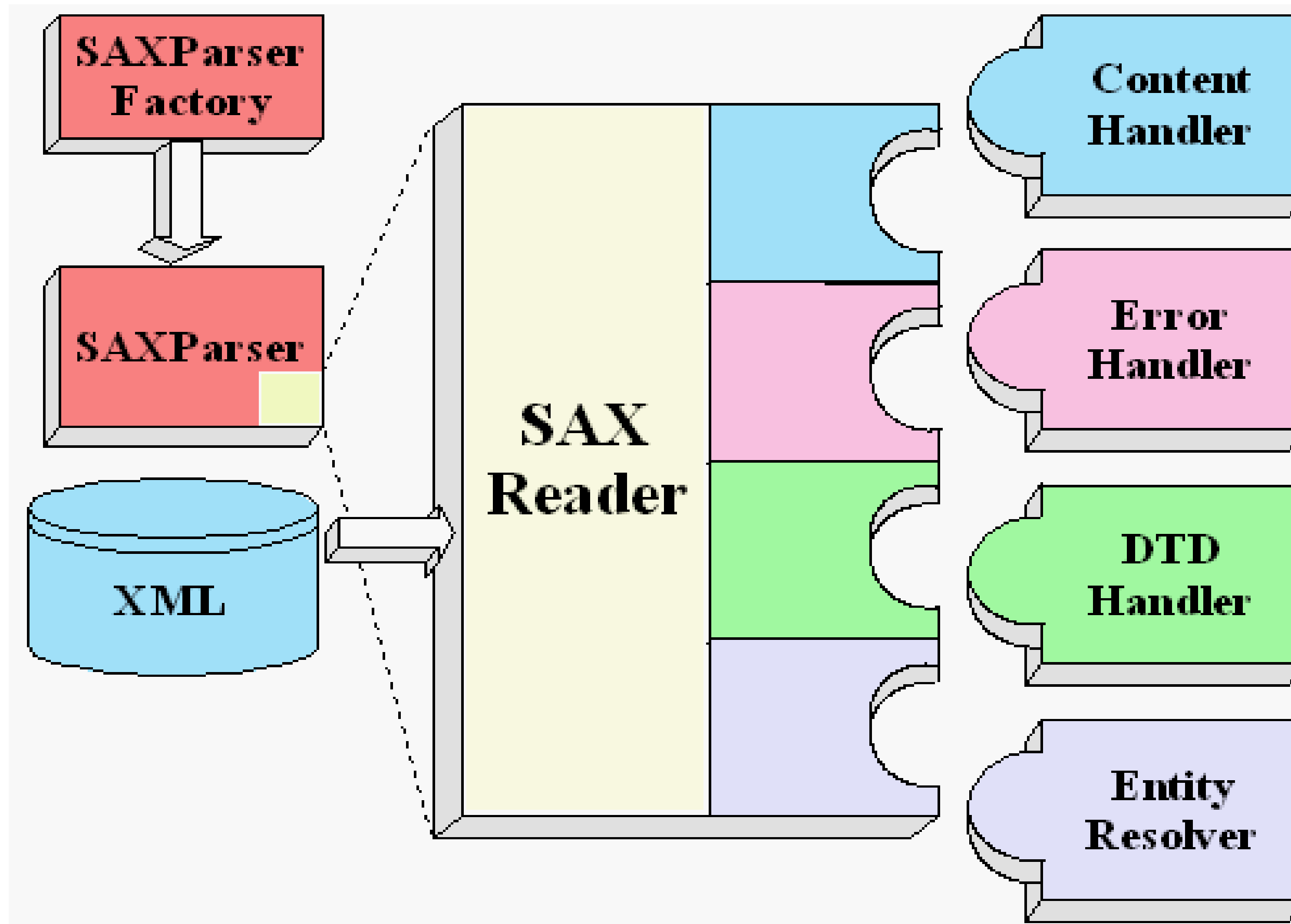
- JAXP – Java API for XML Processing
  - čtení, zápis a transformaci XML
  - SAX, DOM, XSLT
    - podle W3C
  - podporuje různé implementace
    - referenční implementace součástí JDK
      - lze použít jiné
- JDOM
  - http://www.jdom.org/
  - „zjednodušený“ DOM pro Javu
- JAXB – Java Architecture for XML Binding
  - mapování XML <=> Java objekty

- Elliotte Rusty Harold: Processing XML with Java
  - http://www.cafeconleche.org/books/xmljava/
  - kniha – volně ke stažení

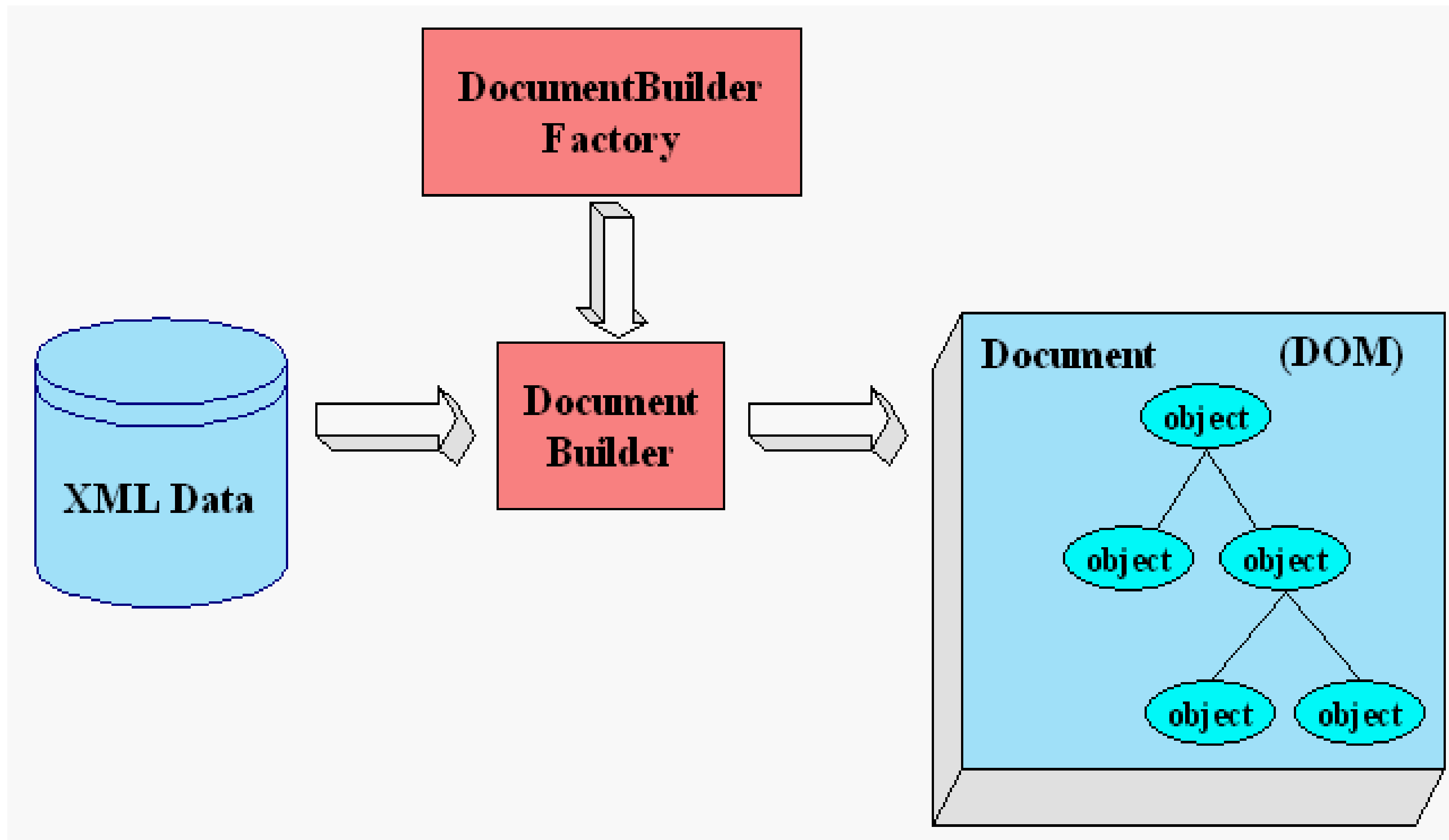
JAXP – přehled

- balíky
  - javax.xml.parsers
  - org.w3c.dom
  - org.xml.sax
  - javax.xml.transform
- SAX (Simple API for XML)
  - průchod přes XML dokument element po elementu
  - na každém elementu něco provést
  - rychlé, nenáročné na paměť
  - složitější na použití
- DOM
  - postaví z dokumentu strom v paměti
  - jednoduché na použití
  - pomalé, náročné na paměť

SAX



DOM



DOM: použití

```
 DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();

// vytvoří celý strom v paměti
Document document = builder.parse("file.xml");

Element root = document.getDocumentElement();
NodeList nl = root.getChildNodes();
for (int i=0; i<nl.length(); i++) {
  Node n = nl.item(i);
  ...
}
```

SAX: použití

```
class MyHandler extends DefaultHandler {
  void startDocument() {
    ...
  }
  void endDocument() {
    ...
  }
  void startElement(....) {
    ...
  }
  ...
}

SAXParserFactory factory =
  SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
saxParser.parse("file.xml", new MyHandler() );
```

Implementace

- existují různé implementace JAXP
- DocumentBuilderFactory.newInstance() i SAXParserFactory.newInstance()
  - uvnitř používají ServiceLoader
  - varianta newInstance(String factoryClassName, ClassLoader classLoader)
    - hledá danou třídu

JDOM – Přehled

- http://www.jdom.org/
- API pro XML
- přímo pro Javu
  - používá std. API z Javy (kolekce,...)
- jednoduché na používání
- rychlé
- "light-weight"

Použití

```
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(filename);
Element root = doc.getRootElement();

List children = current.getChildren();
Iterator iterator = children.iterator();
while (iterator.hasNext()) {
  Element child = (Element) iterator.next();
  ...
}
```

JAVA

JDBC

Přehled

- rozhraní pro přístup k relační databázi
- jednotné
  - nezávislé na databázi
    - výrobce databáze musí dodat JDBC driver
- umožňuje
  - vykonávání SQL dotazů
  - přístup k výsledkům dotazů
    - podobné reflection API
- balíky
  - java.sql, javax.sql

JDBC Driver

- JDBC API
  - v podstatě jen rozhraní
  - implementace dodána přes driver
- driver
  - explicitně natáhnout a zaregistrovat
  - Class.forName("com.driver.Name");
- po natažení driveru, vytvořit připojení na DB
  - Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");
  - url
    - jdbc:mysql://localhost/test
    - jdbc:odbc:source

Základní třídy a rozhraní

- DriverManager – třída
  - všechny metody jsou statické
  - getConnection()
    - několik variant
  - getDrivers()
    - všechny natažené drivery
  - getLogWriter(), setLogWriter()
  - println()
    - zapis do logu
  - getLoginTimeout(), setLoginTimeout()

Základní třídy a rozhraní

- Connection – interface
  - vytváření a vykonávání dotazů
- ResultSet – interface
  - výsledky dotazu

Základní příklad

```
Class.forName("com.mysql.cj.jdbc.Driver");
Connection con = DriverManager.getConnection(
              "jdbc:mysql://localhost/test", "","");

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM
  test");

while (rs.next()) {
  // zpracování výsledků po řádcích
}

stmt.close();
con.close();
```

Přístup k výsledkům

- podbné reflection API
  - getString(), getInt(),...
  - pracuje se nad aktuálním řádkem
  - identifikace sloupce pomocí
    - jména
    - pořadí

```
ResultSet rs = stmt.executeQuery("SELECT ID,
                                  NAME FROM TEST");
while (rs.next()) {
  int id = rs.getInt("ID");
  String s = rs.getString("STRING");
  System.out.println(id + " " + s);
}
```

Přístup k výsledkům

- ResultSet.next()
  - musí být zavoláno i na první řádek
- getString()
  - lze volat na „všechny“ typy
    - nelze na nové SQL3 typy
  - automatická konverze na String

Dotazy

- Connection.createStatement()
  - vytvoření dotazu ("prázdného")
- Statement.executeQuery("....")
  - pro dotazy vracející vysledky (SELECT)
  - výsledky přes ResultSet
- Statement.executeUpdate("...")
  - pro dotazy nevracející výsledky
    - UPDATE
    - CREATE TABLE
    - …

PreparedStatement

- PreparedStatement
  - interface
  - dědí od Statement
  - předpřipravený dotaz s parametry
    - vyplní se před použitím
  - metody
    - set*Type*(int index, *type* v)
    - clearParameters()

```
PreparedStatement pstmt =
con.prepareStatement("UPDATE EMPLOYEES SET
                      SALARY = ? WHERE ID = ?");

pstmt.setBigDecimal(1, 153833.00)
pstmt.setInt(2, 110592)
```

Transakce

- implicitně – auto-commit mód
  - *commit* se provede po každé změně
- auto-commit lze zrušit

```
con.setAutoCommit(false);
//
// posloupnost změn
//
con.commit();    // nebo  con.rollBack()
con.setAutoCommit(true);
```

Callable Statements

- pro přístup k uloženým procedurám
- dědí od PreparedStatement
  - nastavení parametrů
    - set*Type*(int index, *type* v)
  - návratový typ nutno zaregistrovat
    - registerOutParameter(int index, int sqlType)
  - formát
    a) {?= call <procedure-name>[<arg1>,<arg2>, …]}
    b) {call <procedure-name>[<arg1>,<arg2>, …]}

```
CallableStatement cs = con.prepareCall("{call
                                        SHOW_SUPPLIERS}");
ResultSet rs = cs.executeQuery();
```

Ošetření chyb

- SQLException
  - a její potomci
  - String getSQLState()
    - definováno X/Open
  - int getErrorCode()
    - specifický pro konkrétní databázi
- varování (warnings)
  - SQLWarning
  - není to výjimka
  - nutno explicitně testovat
    - Statement.getWarnings()
    - SQLWarning.getNextWarning()

Batch update

- zpracování více dotazů najednou
- Statement.addBatch(String sql)
  - přidá dotaz do dávky
- int[] Statement.executeBatch();
  - provede dávku
  - vrací počet ovlivněných řádků pro každý dotaz v dávce

Updatable ResultSet

- implicitní ResultSet nelze měnit, lze se pohybovat pouze vpřed
  - lze změnit při vytváření Statementu

```
Statement stmt = con.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT ...");
```

  - výsledný ResultSet lze měnit, lze se v něm volně pohybovat, nejsou v něm vidět změny od ostatních uživatelů

Objektové databáze

- ne-relační databáze
- ukládaní a vyhledávání objektů
- vlastní přístup bez JDBC

- db4o
- NeoDatis
- ...

- příklad pro NeoDatis

```
Sport sport = new Sport("volley-ball");
ODB odb = ODBFactory.open("test.neodatis");
odb.store(sport);
Objects<Player> players = odb.getObjects(Player.class);
odb.close();
```

- moc se nepoužívají

ORM

- problém s OO databázemi
  - jednoduché na použití
  - nepříliš výkonné, nepříliš podporované,...
- řešení – ORM
  - (object-relational mapping)
  - vrstva mapující relační databázi na objekty
  - zjednodušeně
    - třída ~ schéma tabulku
    - objekt ~ řádek v tabulce
  - JDBC se typicky používá uvnitř
    - automaticky
  - Hibernate
    - http://hibernate.org/
    - nejpoužívanější ORM pro Javu
      - implementace i pro další technologie

Dokumentové databáze

- ukládání dokumentů
  - semi-structured date
- MongoDB
  - https://www.mongodb.com/
  - dokumenty ~ JSON

```
MongoClient mongoClient = new MongoClient();
MongoDatabase database = mongoClient.getDatabase("mydb");
MongoCollection<Document> collection =
database.getCollection("test");
Document doc = new Document("name", "MongoDB")
                .append("type", "database")
                .append("count", 1)
                .append("versions", Arrays.asList("v3.2",
                                                  "v3.0", "v2.6"))
                .append("info", new Document("x",
                                  203).append("y", 102));
collection.insertOne(doc);
```

Mongo

- existuje i driver pro Mongo pro JDBC
  - collection ~ tabulka

Verze prezentace AJ08.cz.2020.01
Tato prezentace podléhá licenci Creative Commons Uveďte autora-Neužívejte komerčně 4.0 Mezinárodní License.