Web- og serverprogrammering

EJB - dag 9

EJB (Enterprise Java Beans)
Entitetsbønner til databasekommunikation
Evt.: EJBQL (EJB Query Language)
Evt.: EJB: Transaktioner og sikkerhed
Læsning: WJSP kapitel 12

Java 2 Enterprise Edition (J2EE) - til store servere

- J2EE-platformens dele
  - Webserver
  - EJB (Enterprise JavaBeans)
  - CORBA - fjernkald af objekter (ikke kun Java, sværere end RMI)
  - JTS (Java Transaction Service) - transaktioner (commit, rollback)
  - JMS (Java Message Service) - til MQ-systemer.
    - tjeneste til at sende og modtage synkrone eller asynkrone meddelelser mellem programmer
  - JavaMail, XML-behandling, ...

- Sun varetager 'kun' J2EE-*specifikationen*
  - Mange uafhængige udbydere af J2EE-platforme
    - Borland AppServer
    - Oracle
    - IBM WebbSphere
    - JBoss (med åben kildekode)
    - Sun (referenceimplementation)
    - ... (>10 andre)

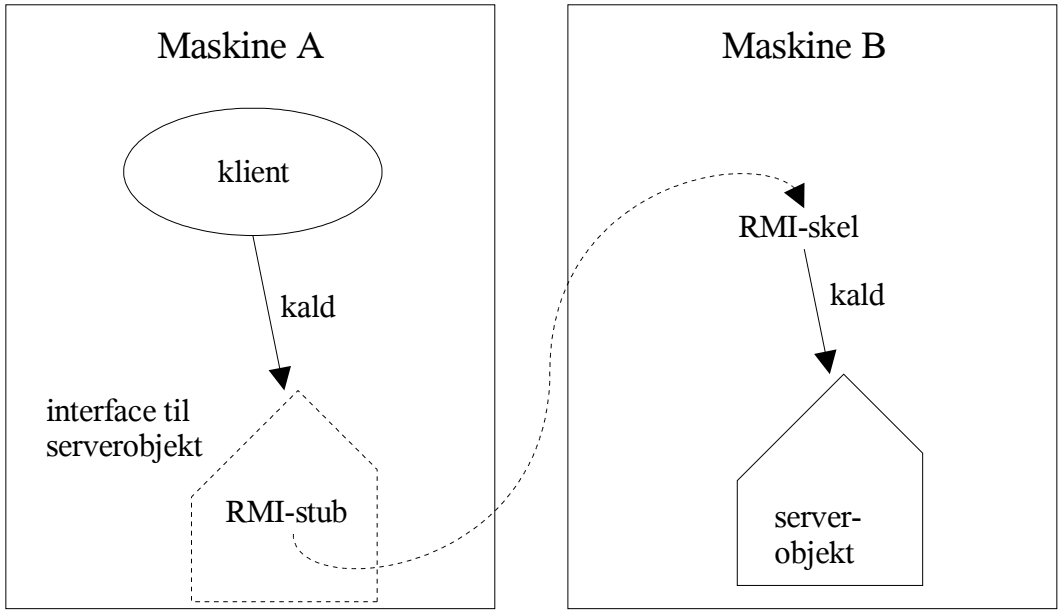
J2EE != udviklingsplatform
Mange udbydere har også egen, 'strømlinet' webudviklingsmodel

Enterprise JavaBeans (EJB) - objekter i serveren

- J2EE-container varetager EJB-objekter
  - Persistens, transaktioner, sikkerhed
  - Kald kan ske lokalt eller over netværket
    - F.eks. fra JSP-sider!

- EJB koncentrerer sig om programlogik
  - Serveren sørger for al 'benarbejdet'
  - Sessionsbønner repræsenterer et forløb
  - Entitetsbønner repræsenterer data fra en tabel
    - Containerstyret persistens (CMP)
      - Al databasekode bliver genereret af containeren (!)
    - Bønnestyret persistens (BMP)
      - Programmøren skriver selv databasekoden
  - Meddelelses-drevne bønner

Principippet i kald af metoder i objekter over netværket



En EJB-bønne kører en en J2EE-applikationsserver - muligvis adskilt fra klienten

Hver EJB-bønne udgøres af tre stykker kildetekst (to interfaces og en klasse):

1) Et **fjerninterface** (eng.: remote interface), der specificerer hvordan bønnen kan bruges af klienten.
2) Et **hjemmeinterface** (eng.: home interface), der specificerer, hvordan bønnen kan fremfindes, oprettes og nedlægges af klienten.
3) En **implementationsklasse** af bønnen, med den programkode, der beskriver, hvad der skal ske, når de forskellige metoder i fjerninterfacet kaldes.

Kildekoden i en EJB

Implementationen af bønnen

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.*;

public class VekslerBean implements SessionBean
{
   public void ejbCreate() {}
   public void setSessionContext(SessionContext sc) {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void ejbRemove() {}

   public double euroTilDollar(int euro) throws RemoteException
   {
      return euro/1.1;
   }

   public double dollarTilEuro(double dollar) throws RemoteException
   {
      return dollar*1.1;
   }
}
```

Fjerninterfacet (hvordan objektet kan bruges af klienten):

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Veksler extends EJBObject
{
  public double dollarTilEuro(double dollar)
     throws RemoteException;

  public double euroTilDollar(int euro)
     throws RemoteException;
}
```

Hjemmeinterfacet (fremfinding)

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface VekslerHome extends EJBHome
{
  public Veksler create() throws RemoteException,
                                 CreateException;
}
```

... plus noget information til EJB-containeren (XML)

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
  <enterprise-beans>
    <session>
      <description>En veksle-bønne (tilstandsløs sessionsbønne)</description>
        <display-name>Veksler</display-name>
        <ejb-name>Veksler</ejb-name>           <!-- JNDI-navn klient bruger -->
        <remote>vp.ejb.Veksler</remote>        <!-- fjerninterfacets navn   -->
        <home>vp.ejb.VekslerHome</home>        <!-- hjemmeinterfacets navn  -->
        <ejb-class>vp.ejb.VekslerBean</ejb-class><!-- klassenavn på bønnen  -->
        <session-type>Stateless</session-type> <!-- bønnen er tilstandsløs  -->
        <transaction-type>Container</transaction-type>
      </session>

<!-- ... beskrivelse af flere bønner kunne komme her ... -->

    </enterprise-beans>
</ejb-jar>
```

Bruge en EJB

Brug af Veksler-bønnen

```
package vp.ejb;
import javax.naming.*;                          // pakken med
JNDI
import javax.rmi.PortableRemoteObject;
public class BenytVeksler
{
  public static void main(String[] args) throws Exception
  {
    // Angiv data til JNDI til dens kontekst.
    // Disse kunne også i stedet angives i jndi.properties

    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
         "com.evermind.server.rmi.RMIInitialContextFactory");
    env.put(Context.SECURITY_PRINCIPAL, "admin");
    env.put(Context.SECURITY_CREDENTIALS, "welcome");
    env.put(Context.PROVIDER_URL,
       "ormi://localhost:23891/current-workspace-app");
    Context ctx = new InitialContext(env);

    VekslerHome hjem = (VekslerHome)ctx.lookup("Veksler");

    Veksler valutaveksler =  hjem.create();

    double beløb = valutaveksler.euroTilDollar(100);
    System.out.println("100 euro er "+beløb+" dollar.");

    beløb = valutaveksler.dollarTilEuro(100);
    System.out.println("100 dollar er "+beløb+" euro.");
  }
}
```

Klienten slår først bønnen op med JNDI og får derved fat i hjemmeinterfacet.
Ved hjælp af dette oprettes (en fjernreference til) bønnen, som derefter anvendes.

Hjemmeinterfacet (fremfinding)

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.*;

public interface VekslerHome extends EJBHome
{
  public Veksler create() throws RemoteException,
                                 CreateException;
}
```

Fjerninterfacet (hvordan objektet kan bruges af klienten):

```
package vp.ejb;
import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Veksler extends EJBObject
{
  public double dollarTilEuro(double dollar)
     throws RemoteException;

  public double euroTilDollar(int euro)
     throws RemoteException;
}
```

Typer af EJB

- Sessionsbønner
  - repræsenterer et eller andet forløb og indeholder de data der knytter sig til dette forløb
    - Tilstandsløse (eksempel: Veksler-bønnen)
      - Flere klienter kan dele samme instans
    - Tilstandsfulde
      - Hver klient får sin egen instans
- Entitetsbønner
  - repræsenterer data fra en tabel. De er *persistente* (dvs. deres data 'bakkes op' af en database).
    - Containerstyret persistens (CMP)
      - Al databasekode bliver genereret af containeren (!)
    - Bønnestyret persistens (BMP)
      - Programmøren skriver selv databasekoden
- Meddelelses-drevne bønner

Entitetsbønner med CMP

Udvikleren designer databasen...

```
CREATE TABLE kunder (navn varchar(32), kredit float)
INSERT INTO kunder VALUES('Jacob', -1799)
INSERT INTO kunder VALUES('Brian', 0)
```

... og beder derefter udviklingsværktøjet generere en Kunde-EJB

Bruge Kunder-objekter fra klienten

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
   "com.evermind.server.rmi.RMIInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "admin");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
env.put(Context.PROVIDER_URL,
   "ormi://localhost:23891/current-workspace-app");
Context ctx = new InitialContext(env);
KundeHome kunderHome = (KundeHome)ctx.lookup("Kunde");
Kunde kunder;

// Use one of the create() methods below to create a new instance
// kunder = kunderHome.create(  );
// kunder = kunderHome.create( java.lang.String navn );

// Retrieve all instances using the findAll() method
// (CMP Entity beans only)
Collection coll = kunderHome.findAll();
Iterator iter = coll.iterator();
while (iter.hasNext())
{
   kunder = (Kunde)iter.next();
   System.out.println("navn = " + kunder.getNavn());
   System.out.println("kredit = " + kunder.getKredit());
   System.out.println();
}
```

Hjemmeinterfacet (fremfinding)

```
public interface KunderHome extends EJBHome
{
  Kunder create()
     throws RemoteException, CreateException;

  Kunder findByPrimaryKey(String primaryKey)
     throws RemoteException, FinderException;

  Collection findAll()
     throws RemoteException, FinderException;
}
```

Fjerninterfacet (hvordan objektet kan bruges af klienten)

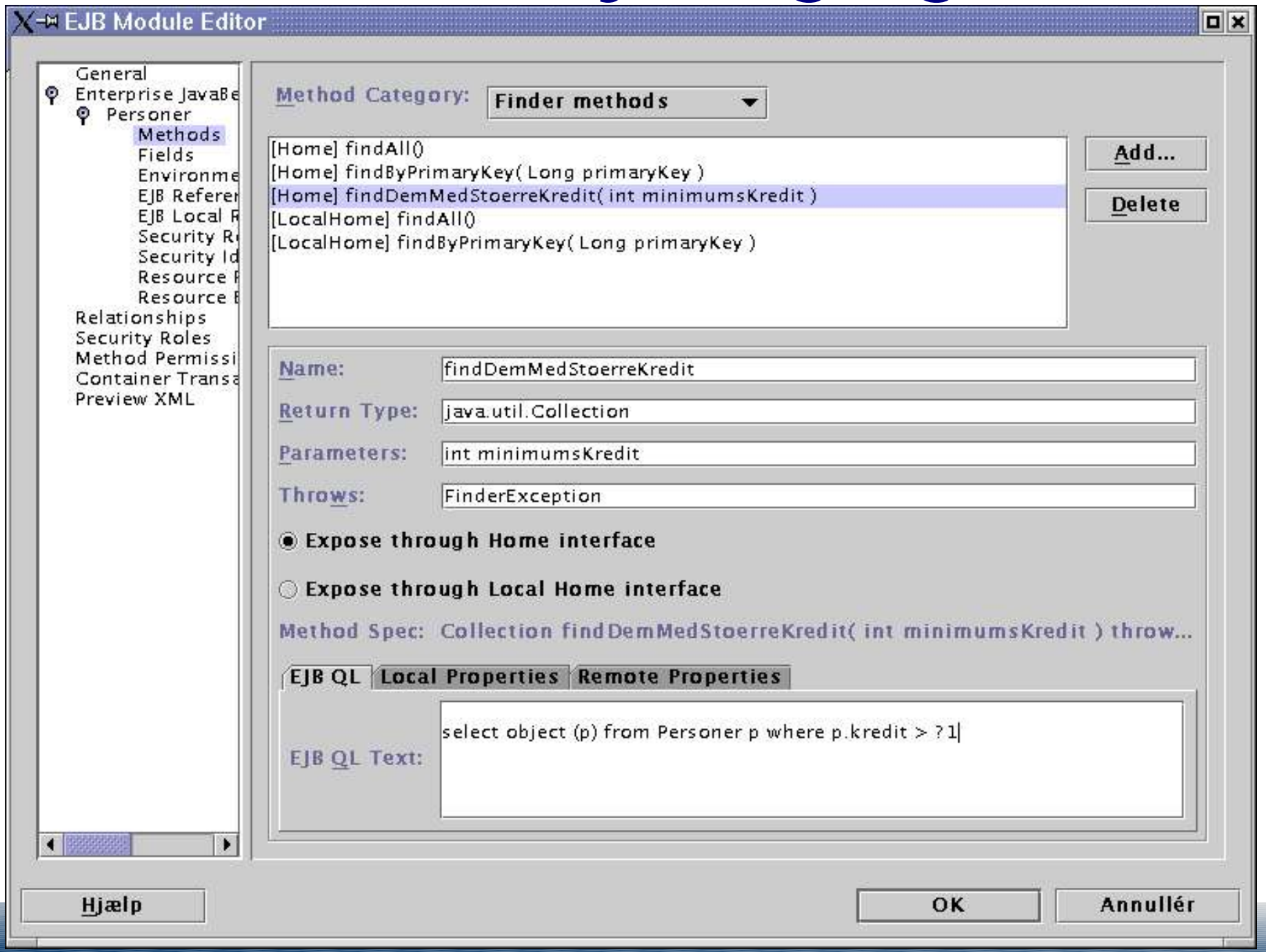
```
public interface Kunder extends EJBObject
{
  String getNavn() throws RemoteException;

  void setNavn(String newNavn) throws RemoteException;

  Float getKredit() throws RemoteException;

  void setKredit(Float newKredit) throws RemoteException;
}
```

EJB Query Language



EJB Query Language

```
    <entity>
      <description>Entity Bean ( CMP )</description>
      <display-name>Personer</display-name>
      <ejb-name>Personer</ejb-name>
      <home>mypackage1.PersonerHome</home>
      <remote>mypackage1.Personer</remote>
      <local-home>mypackage1.PersonerLocalHome</local-home>
      <local>mypackage1.PersonerLocal</local>
      <ejb-class>mypackage1.impl.PersonerBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.Long</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>Personer</abstract-schema-name>
      <cmp-field>
        <field-name>personer_pk</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>navn</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>kredit</field-name>
      </cmp-field>
      <primkey-field>personer_pk</primkey-field>
      <query>
        <query-method>
          <method-name>findDemMedStoerreKredit</method-name>
          <method-params>
            <method-param>int</method-param>
          </method-params>
        </query-method>
        <ejb-ql>select object (p) from Personer p where p.kredit > ?1</ejb-ql>
      </query>
    </entity>
  </enterprise-beans>
</ejb-jar>
```

EJB: Transaktioner

- Programmatisk ('i hånden'-) transaktionsstyring
- Deklarativ transaktionsstyring
  - Metoder markeres med en transaktions-attribut
    - Det er containerens opgave at sørge for at oprette, nedlægge, fuldføre og evt. annullere transaktioner i henhold til disse
  - Required - metoden er med i den aktuelle transaktion.
    Der oprettes en ny transaktion af containeren, hvis der ikke er nogen i gang.
  - Supports - metoden er med i den aktuelle transaktion.
    Der oprettes ikke en ny transaktion af containeren, hvis der ikke er nogen i gang.
  - Mandatory - metoden skal være med en transaktion.
    Er der ikke er nogen i gang, kastes undtagelsen TransactionRequiredException.
  - Never - metoden må aldrig være med en transaktion.
    Er der en transaktion i gang, kastes undtagelsen RemoteException af containeren.
  - RequiresNew - der oprettes en ny transaktion.
    Hvis der er en transaktion i gang, suspenderes denne indtil metoden har afsluttet.
  - NotSupported - kan ikke være med en transaktion.
    Hvis der er en transaktion i gang, suspenderes indtil metoden har afsluttet.

Microsoft .NET

Here's one cut at an itemized list of the technical components making up the .NET platform:

- **C#**, a "new" language for writing classes and components, that integrates elements of C, C++, and Java.
- A "**common language runtime**", which runs bytecodes in an Internal Language (IL) format. Code and objects written in one language can, ostensibly, be compiled into the IL runtime, once an IL compiler is developed for the language.
- A set of **base components**, accessible from the common language runtime, that provide various functions (networking, containers, etc.).
- **ASP+**, a new version of ASP that supports compilation of ASPs into the common language runtime (and therefore writing ASP scripts using any language with an IL binding).
- **Win Forms and Web Forms**, new UI component frameworks accessible from Visual Studio.
- **ADO+**, a new generation of ADO data access components that use XML and SOAP for data interchange.

J2EE og .NET

J2EE

- En standard
  - Flere udbydere (>15)
  - Flere platforme
  - Ét firma styrer (Sun)
- Ét sprog (Java)
- Åben kildekode
- JDBC RowSet
- EJB CMP

.NET

- En samling Windows-produkter
  - Én udbyder
  - Én platform
- Flere sprog
- Lukket
- DataSet
- (mangler)

Microsoft.NET	J2EE	Key differentiators
C# programming language	Java programming language	C# and Java both derive from C and C++. Most significant features (e.g., garbage collection, hierarchical namespaces) are present in both. C# borrows some of the component concepts from JavaBeans (properties/attributes, events, etc.), adds some of its own (like metadata tags), but incorporates these features into the syntax differently. Java runs on any platform with a Java VM. C# only runs in Windows for the foreseeable future. C# is implicitly tied into the IL common language runtime (see below), and is run as just-in-time (JIT) compiled bytecodes or compiled entirely into native code. Java code runs as Java Virtual Machine (VT) bytecodes that are either interpreted in the VM or JIT compiled, or can be compiled entirely into native code.
.NET common components (aka the ".NET Framework SDK")	Java core API	High-level .NET components will include support for distributed access using XML and SOAP (see ADO+ below).
Active Server Pages+ (ASP+)	Java ServerPages (JSP)	ASP+ will use Visual Basic, C#, and possibly other languages for code snippets. All get compiled into native code through the common language runtime (as opposed to being interpreted each time, like ASPs). JSPs use Java code (snippets, or JavaBean references), compiled into Java bytecodes (either on-demand or batch-compiled, depending on the JSP implementation).
IL Common Language Runtime	Java Virtual Machine and CORBA IDL and ORB	.NET common language runtime allows code in multiple languages to use a shared set of components, on Windows. Underlies nearly all of .NET framework (common components, ASP+, etc.). Java's Virtual Machine spec allows Java bytecodes to run on any platform. CORBA allows code in multiple languages to use a shared set of objects, on any platform with an ORB available. Not nearly as tightly integrated into J2EE framework.
Win Forms and Web Forms	Java Swing	Similar web components (e.g., based on JSP) not available in Java standard platform, some proprietary components available through Java IDEs, etc. Win Forms and Web Forms RAD development supported through the MS Visual Studio IDE - no other IDE support announced at this writing. Swing support available in many Java IDEs and tools.
ADO+ and SOAP-based Web Services	JDBC, EJB, JMS and Java XML Libraries (XML4J, JAXP)	ADO+ is built on the premise of XML data interchange (between remote data objects and layers of multi-tier apps) on top of HTTP (AKA, SOAP). .NET's web services in general assume SOAP messaging models. EJB, JDBC, etc. leave the data interchange protocol at the developer's discretion, and operate on top of either HTTP, RMI/JRMP or IIOP.

Åben Dokumentlicens

- Dette foredragsmateriale er under Åben Dokumentlicens (ÅDL)
  - Du har derfor lov til frit at kopiere dette værk
  - Bruger du dele af værket i et nyt værk, skal de dele, der stammer fra dette værk, igen frigives under ÅDL
  - Den fulde licens kan ses på http://www.sslug.dk/linuxbog/licens.html