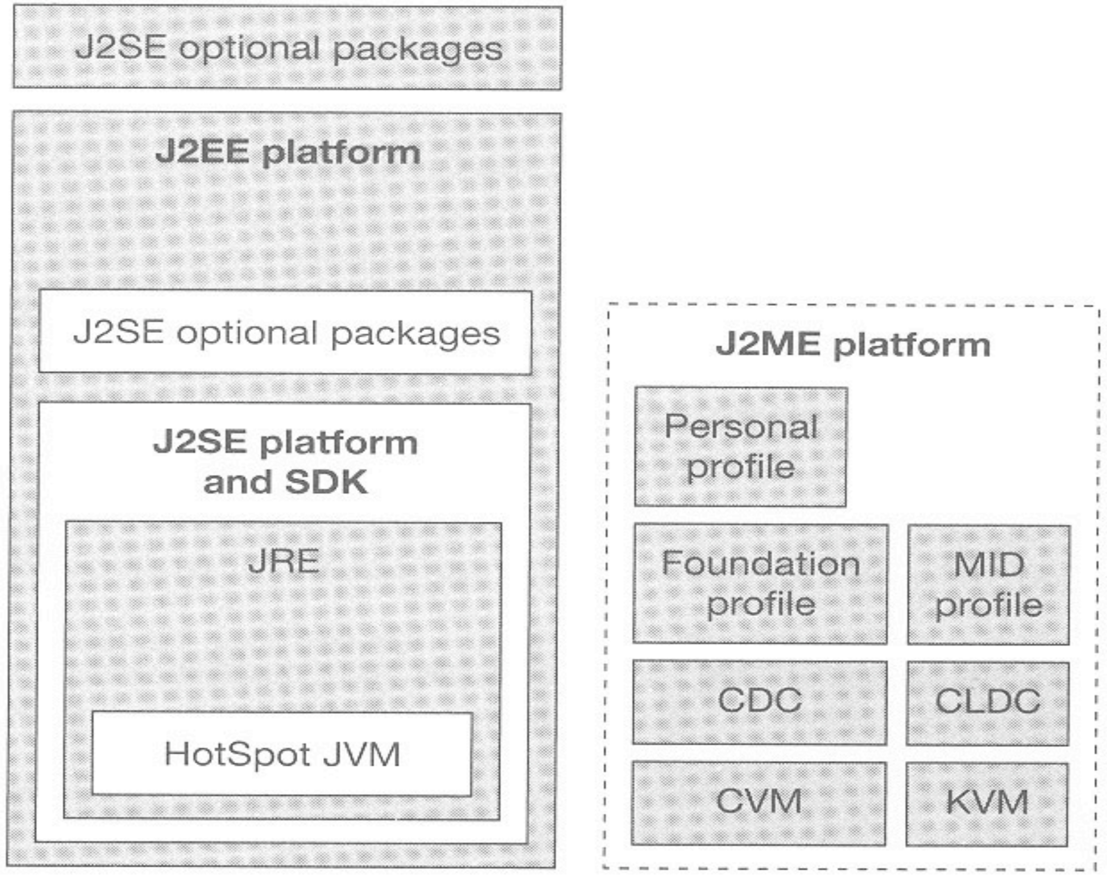
# Java, JavaBeans, EJB (Chapter 14)

Prof. Dr. Wolfgang Pree

Department of Computer Science
University of Salzburg
cs.uni-salzburg.at

## The Sun way – Java, JavaBeans, EJB, and Java 2 editions

- Java is a true phenomenon in the industry.

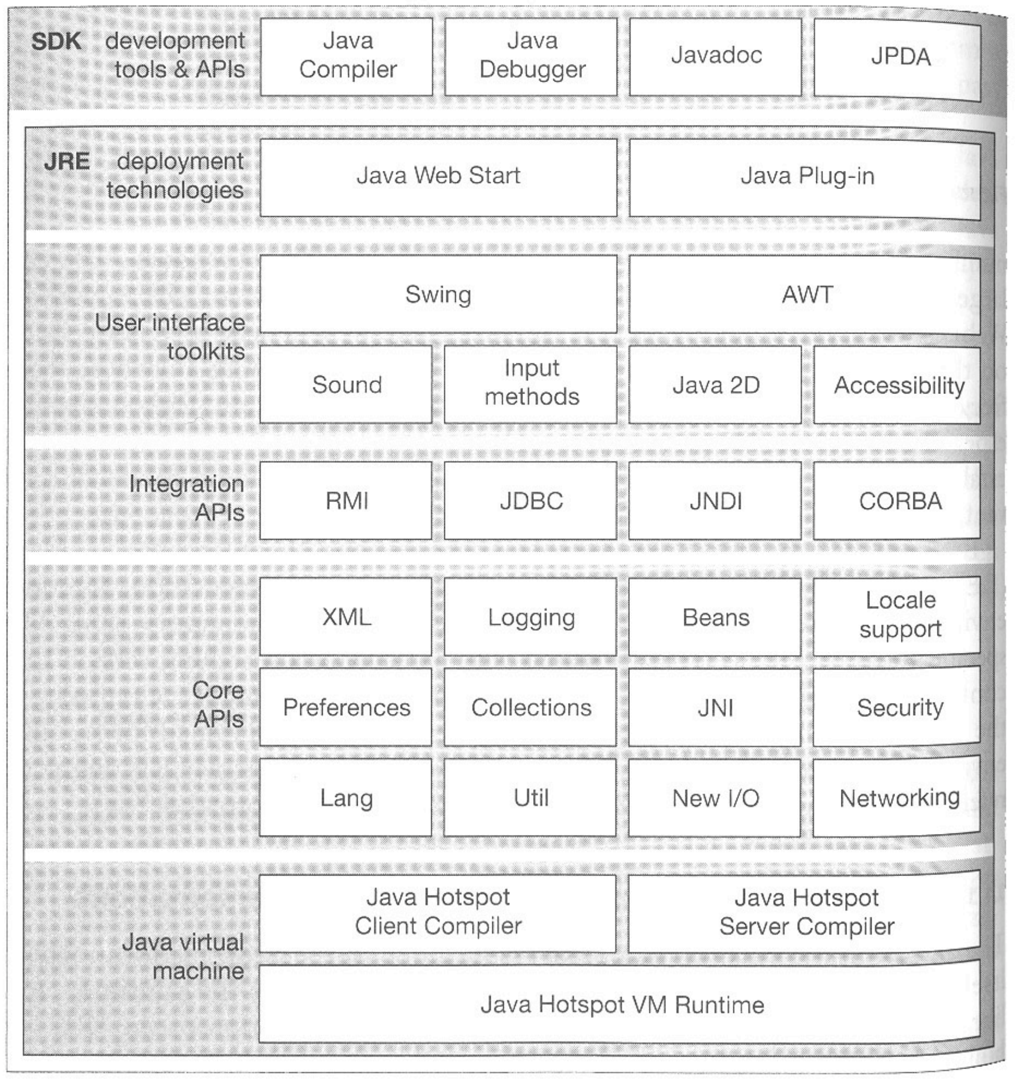


UNIVERSITÄT SALZBURG

## Overview and history of Java component technologies

- The applet was the initial factor of attraction and breakthrough for Java.
- Java was designed to allow a compiler to check an applet’s code for safety.
- Java was designed to allow for the compilation to efficient executables at the target site. Just-in-time (JIT) compilers.
- The second winning aspect of Java is the Java virtual machine (JVM).
- By implementing the JVM on all relevant platforms, Java packages compiled to byte code are platform independent.

# Organization of the Java 2 platform

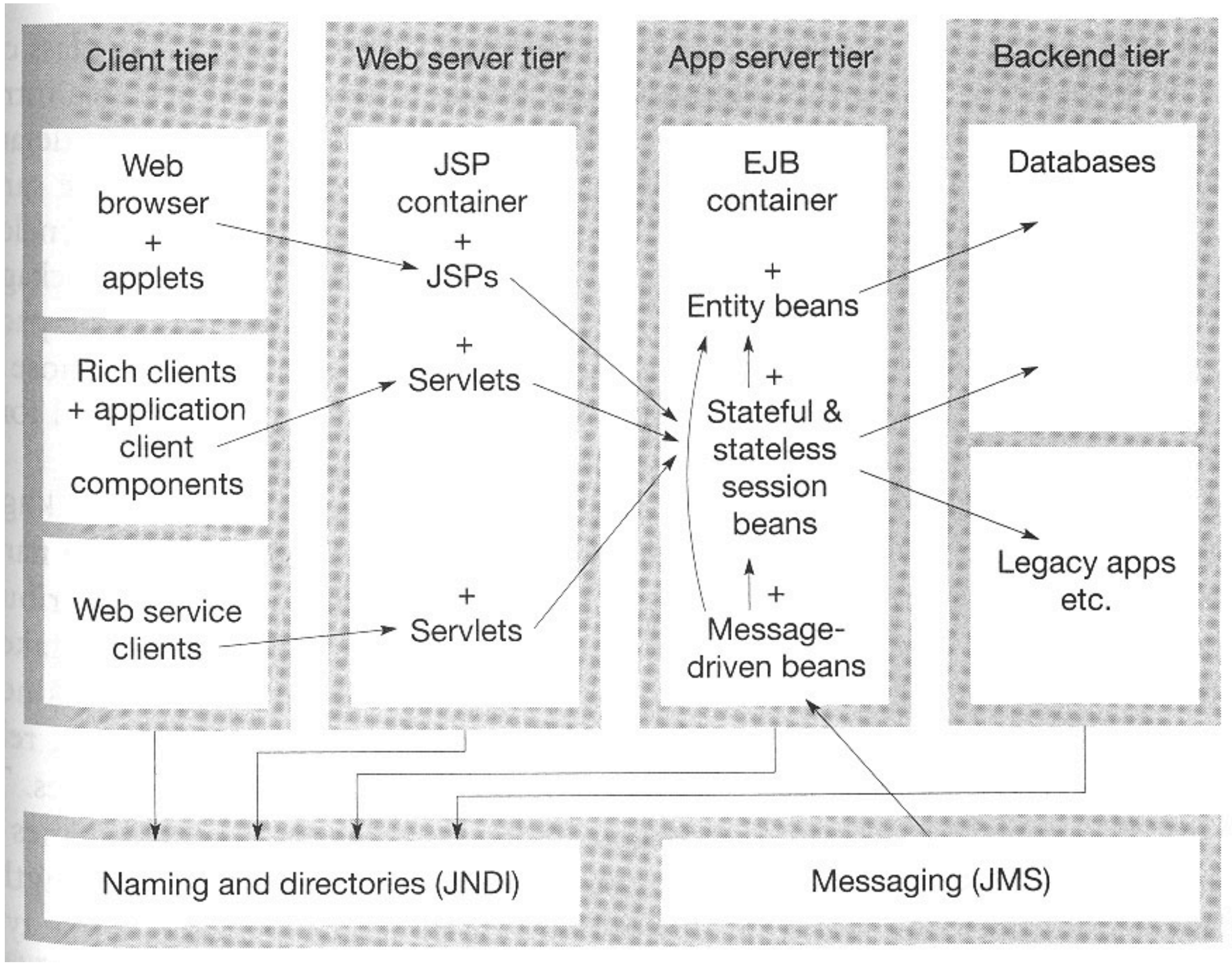


UNIVERSITÄT SALZBURG

# Java 2 Enterprise Edition (J2EE)

- At the heart of the J2EE architecture is a family of component models.
  - on the client side, application components, JavaBeans, and applets
  - on the web server tier, servlets and JSPs
  - On the application server tier, EJB in four variations (stateless session, stateful session, entity, and message-driven beans.
- JavaBeans components aren‘t really confined to any particular tier. Its core technologies could be used in almost any of the spaces shown in the figure.

# Architectural overview of J2EE



UNIVERSITÄT SALZBURG

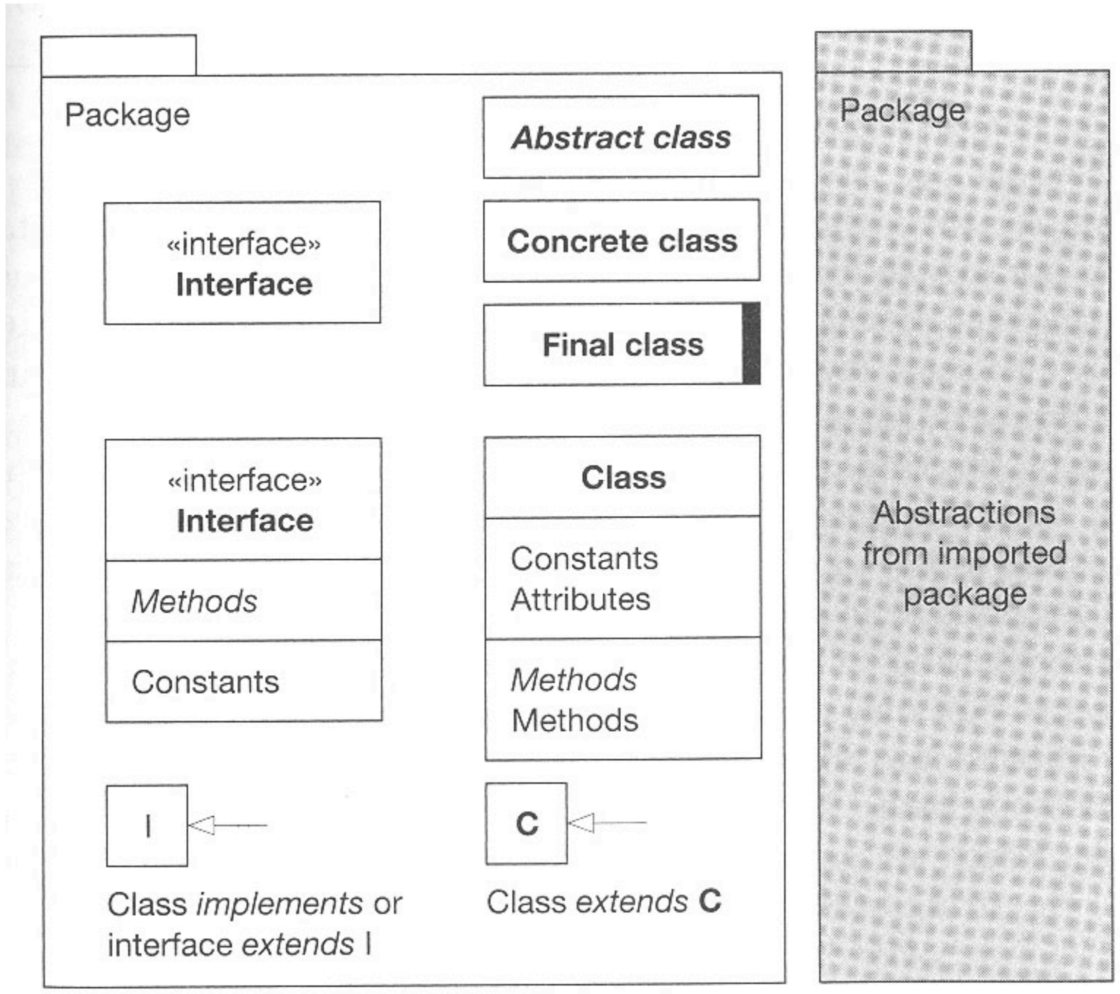
# Java, the language (1)

- Java is an almost pure object-oriented language.
- All Java code resides in methods of classes. All state resides in attributes of class.
- All classes except Object inherit interface and implementation from exactly one other class.
- Objects are either instances of classes or arrays. Objects can be created but not explicitly deallocated.
- All Java classes and interfaces belong to packages.
- Packages introduce a level of encapsulation on top of that introduced by classes. Default mode for access protection allows arbitrary access classes in the same package.

## Java, the language (2)

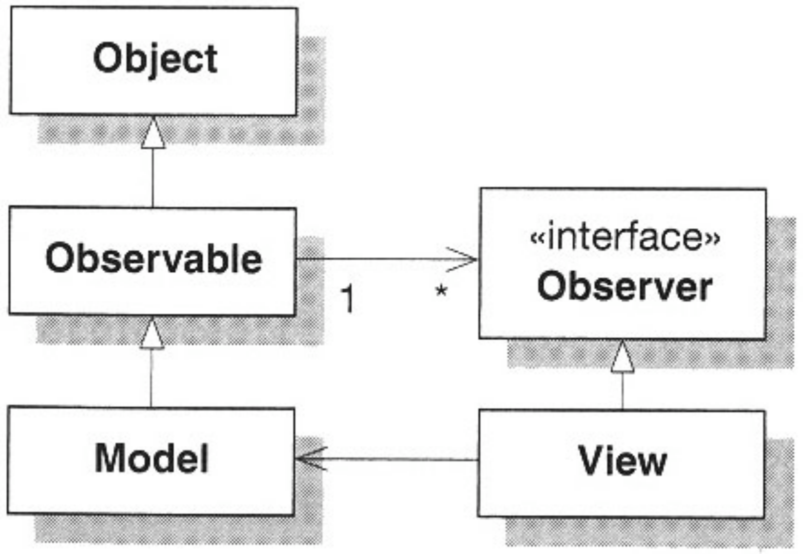
- Interfaces extend any number of other interfaces.
- Classes extend exactly one other class, except for class java.lang.Object, which has no base class.
- Classes can implement any number of interfaces.
- Final methods cannot be overridden.
- Static features belong to the class rather than to instances of the class.
- Interface fields are implicitly public, static, and final making them global constants.
- Interface methods are implicitly public and abstract.

# Java structuring constructs



## Interfaces versus classes

- Separation of interfaces and classes in a way that permits single implementation inheritance combined with multiple interface inheritance.



UNIVERSITÄT SALZBURG

## Exceptions and exception handling

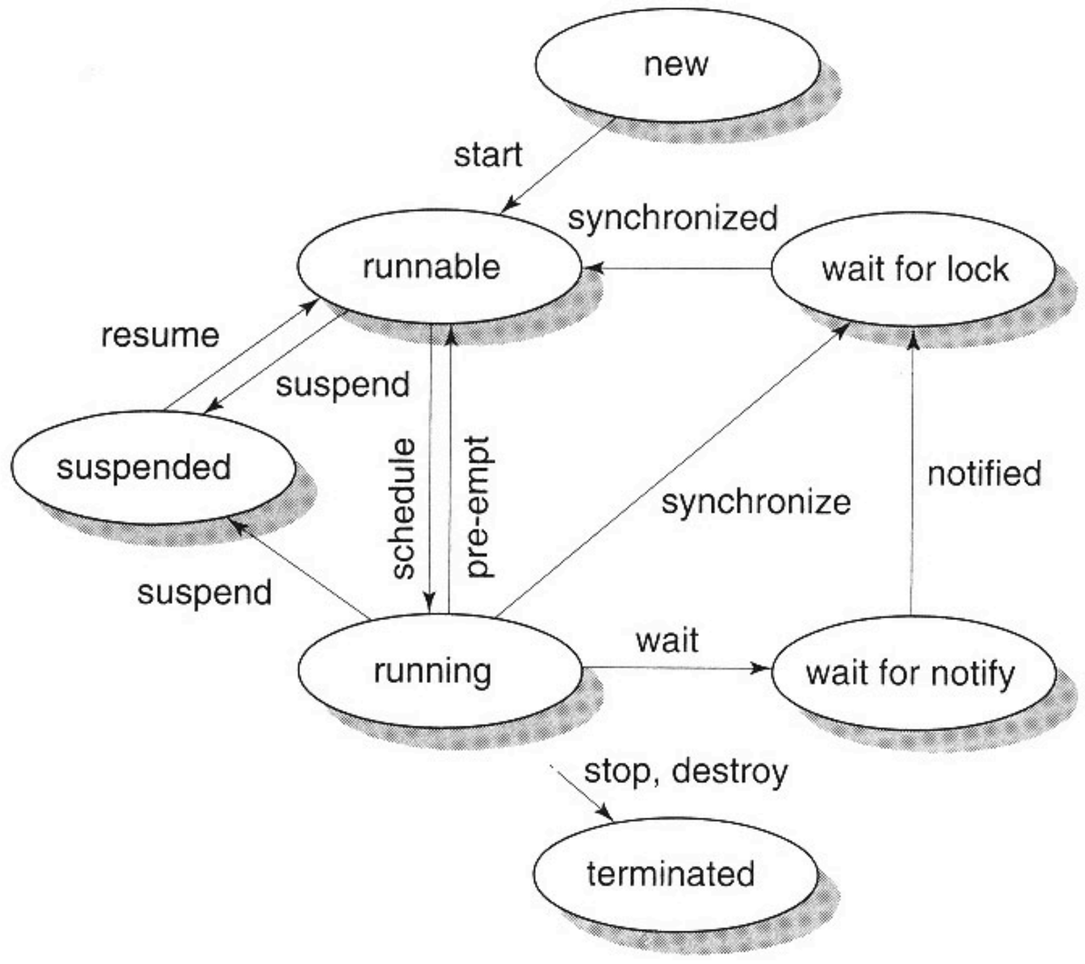
- Exceptions and runtime errors are reflected in Java as exception or error objects. Either explicitly thrown or thrown by the runtime system, such as on out-of-bounds indexing into an array.
- Java has exception types that can only be thrown by a method’s implementation, if the method declaration announced this possibility.

UNIVERSITÄT SALZBURG

## Threads and synchronization

- Java defines a model of concurrency and synchronization and is thus a concurrent object-oriented language. The unit of concurrency is a thread.
- Threads can be dynamically created, pre-empted, suspended, resumed, and terminated.
- Thread termination is not automatically propageted to child threads.
- Java supports thread synchronization either on entrance to a synchronized method or at a synchronized statement.
- Synchronization forces a thread to acquire a lock on an object before proceeding. There is exactly one lock associated with each object.

# States and state transitions of Java threads



## Garbage collection

- Java provides a number of mechanisms to allow programs to interact with the JVM’s garbage collector. The first such mechanism is object finalization.
- The best finalization strategy is to avoid resurrection and to only use finalizers to release external resources.
- A garbage-collection pass can be “encouraged” by calling System.gc. A best-effort finalization pass can be requested by calling *System.runFinalization()*.
- The vast majority of Java classes do not require a finalizer.

## JavaBeans (1)

- The clear distinction between class and object in Java is not carried through in Java Beans. Although a bean really is a component (a set of classes and resources), its customized and connected instances are also called beans.
- Beans have been designed with a dual usage model in mind. Bean instances are first assembled by an assembly tool, such as an application builder, at “design-time,” and are then used a “runtime.”
- A bean instance can customize appearance and functionality dynamically by checking whether or not it is design-time or runtime.

## JavaBeans (2)

- Main aspects of a bean model:
  - Events
  - Properties
  - Introspection
  - Customization
  - Persistence

## Events and connections

- Events in the JDK 1.1 terminology – and as used in JavaBeans – are objects created by an event source and propagated to all currently registered event listeners. Event-based communication generally has multicast semantics.
- A listener can implement a given event listener interface only once. If it is registered with multiple event sources that can all fire the same event, the listener has to determine where an event came from before handling it.
- An event adapter is both an event listener and an event source.

## Properties

- A bean can define a number of properties of arbitrary types. A property is a discrete named attribute that can affect a bean instance's appearance or behavior.
- Typical properties are persistent attributes of a bean instance.
- A property can be bound.
- A property can also be constrained. If not veto exception is thrown, the property is changed in the usual way.
- Normally, properties are edited by property editors. A bean can also nominate a customizer class that implements a specific customization interface.

UNIVERSITÄT SALZBURG

## Introspection

- Events and properties are supported by a combination of new standard interfaces and classes.
- JavaBeans introduces the new notion of method patterns.
- A method pattern is a combination of rules for the formation of a method's signature.
- Method patterns allow for the lightweight classification of individual methods of an interface or class.

## JAR files – packaging of Java components

- JAR files were originally introduced to support JavaBeans, but have since been used to package all other Java components as well.
- The archive may include:
  - A set of class files;
  - A set of serialized objects that is often used for bean prototype instances;
  - Optional help files in HTML;
  - Optional localization information used by the bean to localize itself;
  - Optional icons held in icon files in GIF format;
  - Other resource files needed by the bean.
- The serialized prototype contained in the JAR file allows a bean to be shipped in an initialized default form.

## Reflection

- The Java core reflection service is a combination of original Java language features, a set of support classes and a language feature to support class literals.
- The reflection service allows:.
  - Inspection of classes and interfaces for their fields and methods;
  - Construction of new class instances and new arrays;
  - Access to and modification of fields of objects and classes;
  - Access to and modification of elements of arrays;
  - Invocation of methods on objects and classes.

## Object serialization

- To be serializable, an object has to implement interface *java.io.Serializable*.
- A simple versioning scheme is supported – a serializable class can claim to be a different version of a certain class by declaring a unique serial version unique ID.
- Object serialization must be used with care to avoid security holes.
- Object serialization creates a stream of bytes in a single-pass progress.

## Java native interface

- The Java native interface (JNI) specifies, for each platform, the native calling conventions when interfacing to native code outside the Java virtual machine.
- JNI also specifies how such external code can access Java objects for which references were passed.
- JNI allows native methods to: problems:
  - Create, inspect, and update Java objects;
  - Call Java methods;
  - Catch and throw exception;
  - Load classes and obtain class information;
  - Perform runtime type checking.

# Java AWT and JFC/Swing

- Delegation-based event model;
- Data transfer and clipboard support;
- Drag and drop;
- Java 2D Classes;
- Printing;
- Accessibility;
- Internationalization;
- Swing components and pluggable look and feel.

# Advanced JavaBeans specifications

- **Containment and services protocol**
  supports the concept of logically nesting JavaBeans bean instances.
- A nested bean can acquire additional services at runtime from its container and a container can extend services to its nested bean.
- **Java activation framework (JAF)**
  used to determine the type of arbitrary data, the operations (“commands”) available over that type of data, to clocate, load, and activate components that provide a particular operation over a particular type of data. Command map – a registry for components that is indexed by pairs of MIME types and command names.
- **Long-term persistence for JavaBeans**
  an alternative to object serialization. The file format is XML with a DTD or one or two proprietary Java file formats.
- **InfoBus**
  creates a generic framework for a particular style of composition.The idea is to design beans to be InfoBus-aware and categorize beans into data producers, data consumers, and data controllers, all of which can be coupled by an information bus determining data flow.

## Component variety – applets, servlets, beans, and Enterprise beans (1)

- The Java universe defines five different component models, and more may arrive in the future. The applet and JavaBeans models are Enterprise JavaBeans, servlets, and application client components.
- Applets were the first Java component model, aiming at downloadable lightweight components that would augment websites displayed in a browser.
- The second Java component model, JavaBeans, focuses on supporting connection-oriented programming and is, as  such, useful on both clients and servers.
- JavaBeans are more popular in the scope of client-side rich applications and sometimes perceived as being replaced by EJB on the server.
- JavaBeans remain useful when building client-side applications as they explicitly support a visual application design mode.

UNIVERSITÄT SALZBURG

## Java server pages (JSP) and servlets (1)

- Serving web pages and other formatted contents can be viewed as a combination of three core functions.
- incoming requests for such contents need to be accepted, checked for proper authorization, and routed to the appropriate components ready to handle a particular request.
- Processes to retrieve or synthesize the requested contents.
- The retrieved or generated contents need to be sent to the requesting party.
- The prototypical model handling these three steps is that of a web server.
- A JSP server activates servlets as needed to handle requests.
- It is possible and common to implement servlets instead of writing JSP pages.

# Java server pages (JSP) and servlets (2)

- The servlet programming model is quite natural if relatively small static HTML (or other markup) fragments need to be combined with computed results.
- The dual model to embedding markup in source code is to embed source code in markup
  - JSP page
- To keep JSP pages largely contents-oriented and maintain both readability and localizability, it is useful to minimize code in JSP pages.

## Contextual composition – Enterprise JavaBeans (EJB) (1)

- JavaBeans approach to composition is connection-oriented programming or, as wiring. Beans can define both event sources and event listeners.
- Java Beans model introduce support for hierarchical container structures.
- The InfoBus allows for a flexible decoupling of event sources and event listeners by routing some or all communication through a bus structure that allows for the interception of messages and application of policies without requiring cooperation from the event source or listener and without a need to re-wire.
- EJB. There are no provisions for connection-oriented programming at all.
- EJB components follow a relatively conventional model of object-oriented composition.
- EJB is not about systematically improving compositionality of e-beans via wiring of connections.
- EJB is weak at the connection-oriented composition level, it is strong at the level of contextual composition.
- An EJB container configures services to match the needs of contained beans.

## Contextual composition – Enterprise JavaBeans (EJB) (2)

- The container can use declared relationships to automatically find the entity bean instance at the other end of a relationship of an entity bean instance.
- Deployed beans are conceptually composed with services and resources by an EJB container.
- To enable interaction between services and an instance, contextual access to services is provided to the instance – thus contextual composition.
- All access to beans is required to go through one of two interfaces – the EJB home interface for lifecycle operations and the equivalent of static methods and the EJB object interface for all methods on the bean instance. Non-local clients will see these interfaces implemented on stub objects that use RMI or RMI-over-IIOP to communicate with the corresponding EJB container, which then relays calls to bean instances it contains. Local clients can request local versions of both home and object interfaces. Not allowed to access other beans directly.
- The EJB specification does not detail how a particular container wraps bean instances.

## Beans of many flavors (1)

- Four kinds of EJB beans:
  - stateless session;
  - stateful session;
  - entity and
  - message-driven beans.

## Beans of many flavors (2)

- **Session beans**
  is created by a client as a session-specific contract point.
- A stateless session bean does not maintain state across multiple invocations.
- A stateful session bean remains associated with the one session for its lifetime and thus can retain state across method invocations.
- **Entity beans**
  use objects corresponding to database entities and encapsulate access to actual database records.
- **Entity relationships and database mapping**
  combining the descriptors for relationship among entity beans and those for container-managed persistent fields yields an abstraction for a flexible object-to-relational mapping.
- EJB 2.0 supports one-to-one, one-to-many, and many-to-many relation-ships – all in both unidirectional and bidirectional versions.

## Data-driven composition – message-driven beans in EJB 2.0

- Support for data-driven composition is done by adding an entirely new e-bean type – message-driven beans (md-beans).
- They have no container-managed persistent state.
- Don’t have a remote or local interface or home interface. The only way to instantiate and use an md-bean is to register it for a particular message queue or topic as defined by the Java message service.
- A md-bean can be registered with exactly one such queue or topic only, requiring external collection of messages meant to be handled by an md-bean into a single queue or topic.
- It is possible to write all application logic using md-beans.

UNIVERSITÄT SALZBURG

# Distributed object model and RMI

- Distributed computing is mainly supported by the object serialization service and the remote method invocation (RMI) service.
- A distributed object is handled via references of interface type – it is not possible to refer to a remote object's class or any of its superclasses.
- A remote operation can always fail as a result of network or remote hardware problems.
- Parameter passing. If an argument is of a remote interface type, then the reference will be passed. In all other cases, passing is by value.
- The Java distribution model extends garbage collection as well. Based on a careful bookkeeping of which objects may have remote references to them.
- Java RMI interferes with the notion of object identity in Java with the Java locking system's semantics, which normally prevents self-inflicted deadlocks.

# Java and CORBA

- Important reason to incorporate CORBA into Java projects is to enable the use of IIOP for communication with non-Java subsystems.
- RMI is normally implemented using a proprietary protocol which limits the use of RMI to Java-to-Java communication.
- RMI-over-IIOP does not support the RMI distributed garbage collection model and thus falls back on CORBA’s lifecycle management approach to deal with the lifetime of remote objects explicitly.

UNIVERSITÄT SALZBURG

## Enterprise service interfaces (1)

- **Java naming and directory interface (JNDI)**
  Location of services by exact name or attributes.
  - naming services
  - Directory services
  - DNS
  - The RMI registry, and
  - The CORBA naming service.
  - LDAP-compliant directories

## Enterprise service interfaces (2)

- **Java message service (JMS)**
  JMS is a Java access mechanism to messaging systems – it doesn't implement messaging itself.
- JMS support message queues for point-to-point delivery of messages. It also supports message topics that allow for multiple targets to subscribe. Messages published to a topic are delivered to all subscribers of that topic.
- **Java database connectivity (JDBC)**
  JDBC API is split into the core API and the JDBC optional package.
- JDBC depends on drivers to map the JDBC API to the native interfaces of a particular database.
- **Java transaction API and service (JTA, JTS)**
  Transaction management is almost always delegated to an EJB's container, there are cases where explicit transaction management is required.
- Java transaction  API comprises a low-level XA interface used by server/container implementations and a high-level client interface accessible to EJB beans implementations.

## Enterprise service interfaces (3)

- **J2EE connector architecture (JCA)**
  standardizes connectors between a J2EE application server and enterprise information systems (EIS) such as database management, enterprise resource planning (ERP), enterprise asset management (EAM), and customer relationship management (CRM) systems.
- defines resource adapters that plug into a J2EE application server – one such adapter per EIS type.

## Enterprise service interfaces (4)

- **Java and XML**
  Java architecture for XML Binding (JAXB) provides an API and tools that automate the mapping between XML documents and Java objects.
- Java API for XML messaging (JAXM) is a J2SE optional package that implements the simple objects access protocol (SOAP), v1.1 with attachments.
- Java API for XML-based RPC (JAX-RPC) supports the construction of web services and clients that interact using SOAP and that are described using WSDL.
- Java API for XML processing (JAXP) is a collection of DOM, SAX, and XSLT implementations.
- Java API for XML registries (JAXR) provides uniform and standard access to different kinds of XML registries, including the ebXML registry and repository standard and the UDDI specification.

## Interfaces versus classes in Java, revisited

- Java separates and supports classes and interfaces, where an interface is essentially a fully abstract class.
- JavaBeans allows classes to surface on bean boundaries.
- The Java AWT event model has been changed from an inheritance-based solution in JDK 1.0 to a “delegation-based” (really a forwarding-based) solution in JDK 1.1.
- The 1.0 approach led to undue complexity and was a source of subtle errors.
- The non-distributed Java object model supports classes and interfaces, the distributed Java object model restricts remote access to interfaces.
- Enterprise JavaBeans restricts bean access to interfaces, there can be only one remote interface on a EJB bean. If an EJB bean would need to implement multiple interfaces, it suffices to define a single interface extending these interfaces.

## JXTA and Jini (1)

- JXTA and Jini address a similar problem – the federation of systems over loosely coupled distributed systems.
- Jini focuses on Java everywhere.
- Java-specific networking protocols such as RMI.
- Jini moves Java components to where they are needed.
- JXTA aims at open peer-to-peer computing, preferring XML-based conventions and protocols on the network.
- Jiro aims to use federation to specifically aid systems management.

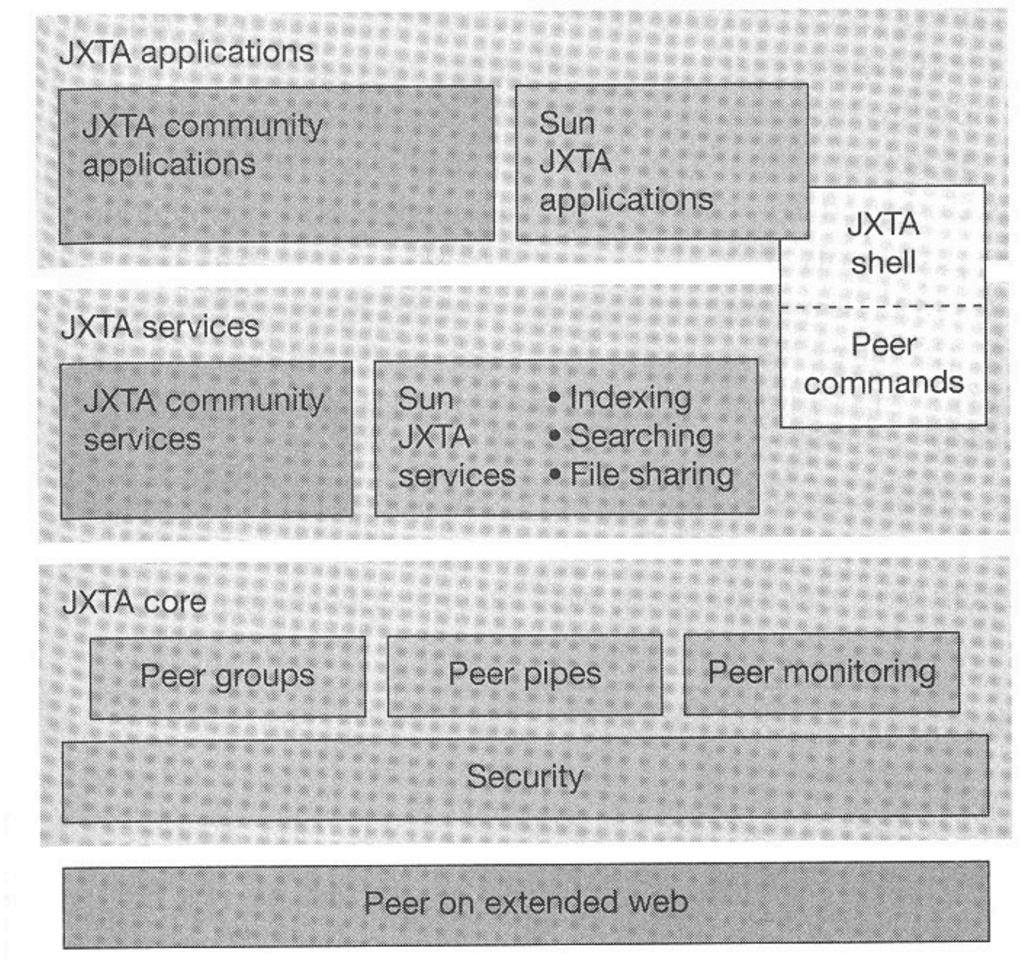
## JXTA and Jini (2)

- **Jini – federations of Java services and clients**
  Jini describes how federations of Java services and clients of such services can be formed over loosely coupled systems.
- How clients locate services, how services make themselves available, and how the overall system copes with partial failure and reconfiguration.
- Jini defines special services called lookup services, clients query the lookup services in order to find specific services, and services publish themselves by registering with lookup services.
- A Jini service can register with more than one lookup service and multiple Jini services can register under the same service type. A Jini client can consult multiple lookup services.
- Once a client has located a service, Jini is out of the loop.

# JXTA and Jini (3)

- **JXTA – peer-to-peer computing**
  JXTA is supposed to span programming languages, platforms (such as Unix versus Windows), and networking infrastructure (such as TCP/IP versus Bluetooth).
- JXTA comprises the following core protocols – peer discovery, peer resolver, peer information, peer membership, pipe binding, and endpoint routing.
- Communication is by means of messages sent via pipes.
- Higher-level communication semantics, such as reliable delivery, can be built on top of basic pipes.
- The peer resolver protocol is used to implement advanced search functionally.
- The peer information protocol can be used to retrieve status (liveness) and capability information on other peers.
- JXTA is a bit like a virtual internet over the internet.

# JXTA architectural layering


JXTA applications
JXTA community applications
Sun JXTA applications
JXTA shell
Peer commands
JXTA services
JXTA community services
Sun JXTA services
• Indexing
• Searching
• File sharing
JXTA core
Peer groups
Peer pipes
Peer monitoring
Security
Peer on extended web