BEANS

What is Beans?

- JavaBeans are software component models. A Java Bean is a general – purpose component model. A Java Bean is a reusable software component that can be visually manipulated in builder tools.
- Their primary goal of a Java Bean is Write Once Run Anywhere.
- Java Beans should adhere to portability, reusability and interoperability.

Bean Properties

- Simple Property:
  - Simple properties are basic, independent, individual properties like width, height, and color.
- Indexed Property:
  - It is a property that can take on an array of values.
- Bound Property:
  - It is a property that alerts other objects when its value changes.
- Constrained Property:
  - It differs from bound property in that it notifies other objects of an impending change. Constrained properties give the notified objects the power to veto a property change.

Introduction of EJB

- Enterprise Java Beans are software component models, their purpose is to build/ support enterprise specific problems.
- EJB – is a reusable server side software component.
- EJB facilitates the development of distributed java applications, providing an object oriented transactional environment for building distributed, multi – tier enterprise components.
- An EJB is a remote object, which needs the services of an EJB container in order to execute.

Continue....

- EJB is an essential part of J2EE platform.
- J2EE platform have component based architecture to provide multi-tiered, distributed and highly transactional features to enterprise level applications.
- EJB provides an architecture to develop and deploy component based enterprise applications considering robustness, high scalability and high performance.
- An EJB application can be deployed on any of the application server compliant with J2EE 1.3 standard specification.

Benefits of EJB

- Simplified development of large scale enterprise level application.
- Application Server/ EJB container provides most of the system level services like transaction handling, logging, load balancing, persistence mechanism, exception handling and so on. Developer has to focus only on business logic of the application.
- EJB container manages life cycle of EJB instances thus developer needs not to worry about when to create/delete ejb objects.

When to Use EJB

- The application must be scalable. To accommodate a growing number of users, you may need to distribute an application's components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.
- Transactions must ensure data integrity. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.

Continue...

- The application will have a variety of clients. With only a few lines of code, remote clients can easily locate enterprise beans. These clients can be thin, various, and numerous.

Types of Bean

Types	Description
Session Bean	Session bean stores data of a particular user for a single session. It can be stateful or stateless. It is less resource intensive as compared to entity beans. Session bean gets destroyed as soon as user session terminates.

Continue...

Types	Description
Entity Bean	Entity beans represents persistent data storage. User data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean.
Message Driven Bean	Message driven beans are used in context of JMS (Java Messaging Service). Message Driven Beans can consumes JMS messages from external entities and act accordingly.

Session Bean

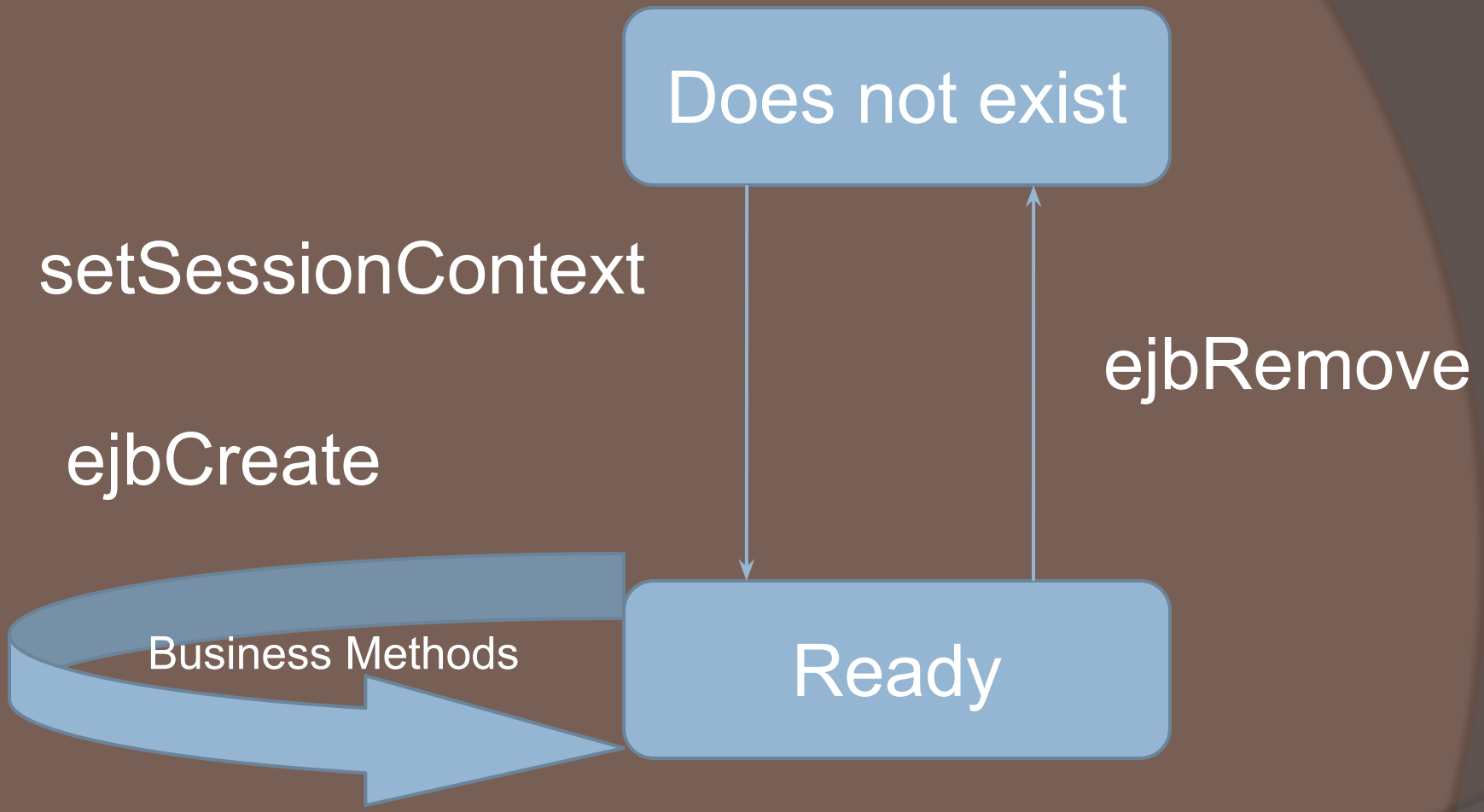
- A session EJB is a non persistent object.
- Its lifetime is the duration of a particular interaction between the client and the EJB.
- The client normally creates an EJB, calls methods on it, and then removes it. If, the client fails to remove it, the EJB container will remove it after a certain period of inactivity.
- There are two types of session beans:

Stateless Session Beans

- A stateless session EJB is shared between a number of clients.
- It does not maintain conversational state.
- After each method call, the container may choose to destroy a stateless session bean, or recreate it, clearing itself out, of all the information pertaining the invocation of the last method.
- The algorithm for creating new instance or instance reuse is container specific.

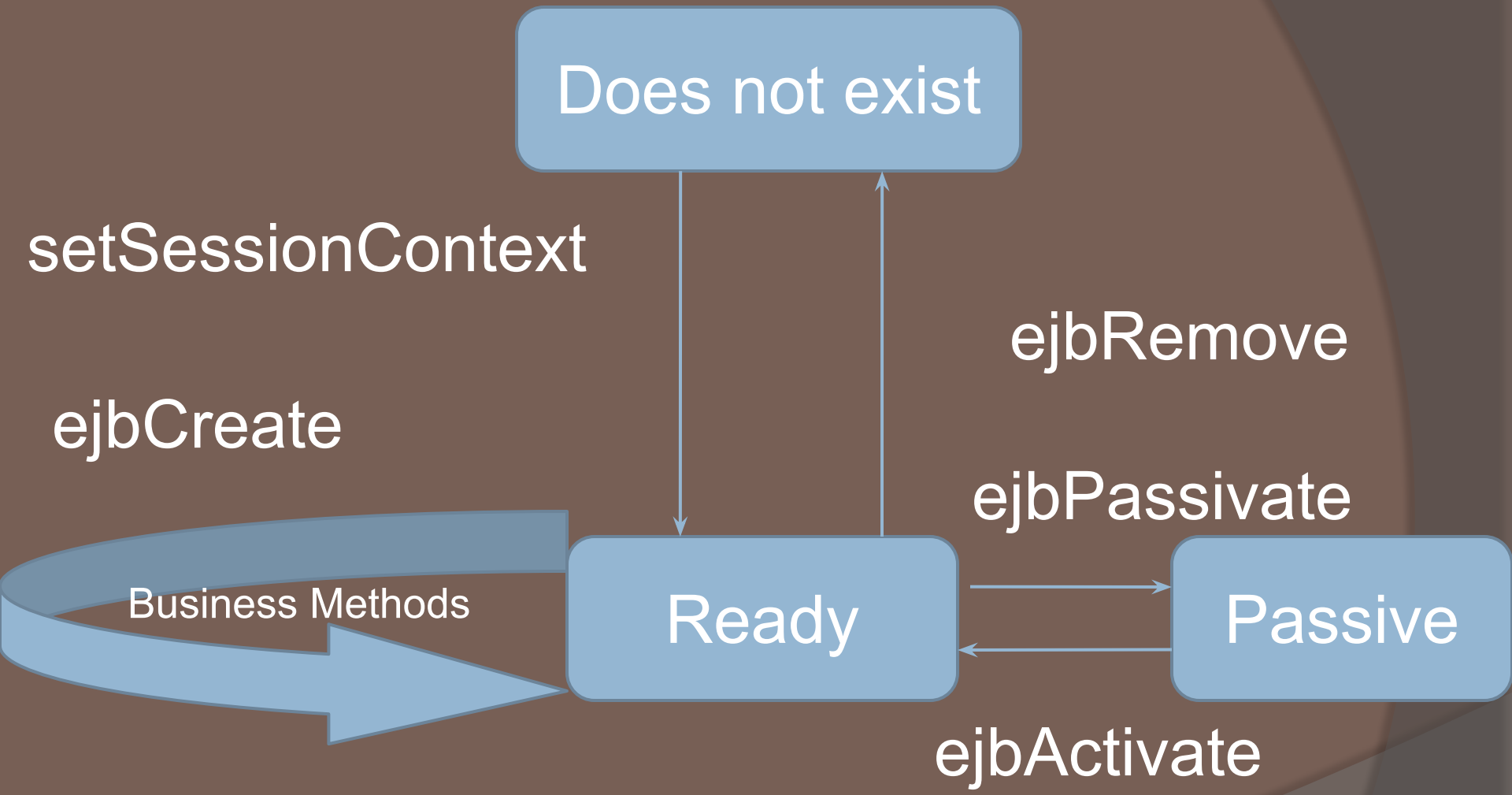
Stateful Session Beans

- A stateful session bean is a bean that is designed to service business process that span multiple method requests or transaction.
- To do this, the stateful bean retains the state for an individual client.
- If, the stateful bean's state is changed during method invocation, then, that same state will be available to the same client upon invocation.



Lifecycle Stateless Session Bean

- Does not exist:
  - In this state, the bean instance simply does not exist.
- Ready state:
  - When EJB server is first started, several bean instances are created and placed in the ready pool. More instances might be created by the container as and when needed by the EJB container.



Lifecycle Stateful Session Bean

- Does not exist:
  - In this state, the bean instance simply does not exist.
- Ready state:
  - A bean instance in the ready state is tied to a particular client and engaged in a conversation.
- Passive state:
  - A bean instance in the passive state is passivated to conserve resources.

Methods of Session Bean

- setSessionContext(SessionContext ctx)
  - Associate your bean with a session context. Your bean can make a query to the context about its current transactional state, and its current security state.
- ejbCreate(…)
  - Initialise your session bean. You would need to define several ejbCreate(…) and then, each method can take up different arguments. There should be at least one ejbCreate() in a session bean.
- ejbPassivate()
  - This method is called for, just before the session bean is passivated and releases any resource that bean might be holding.

- ejbActivate()
  - This method is called just for, before the session bean is activated and acquires the resources that is requires.
- ejbRemove()
  - This method is called for, by the ejb container just before the session bean is removed from the memory.

Entity Bean

- Entity EJBs represent persistent objects.
- Their lifetimes is not related to the duration of interaction with clients.
- In nearly all cases, entity EJBs are synchronized with relational databases.
- This is how persistence is achieved. Entity EJBs are always shared amongst clients.
- A client cannot get an entity EJB to itself. Thus, entity EJBs are nearly always used as a scheme for mapping relational databases into object – oriented applications.

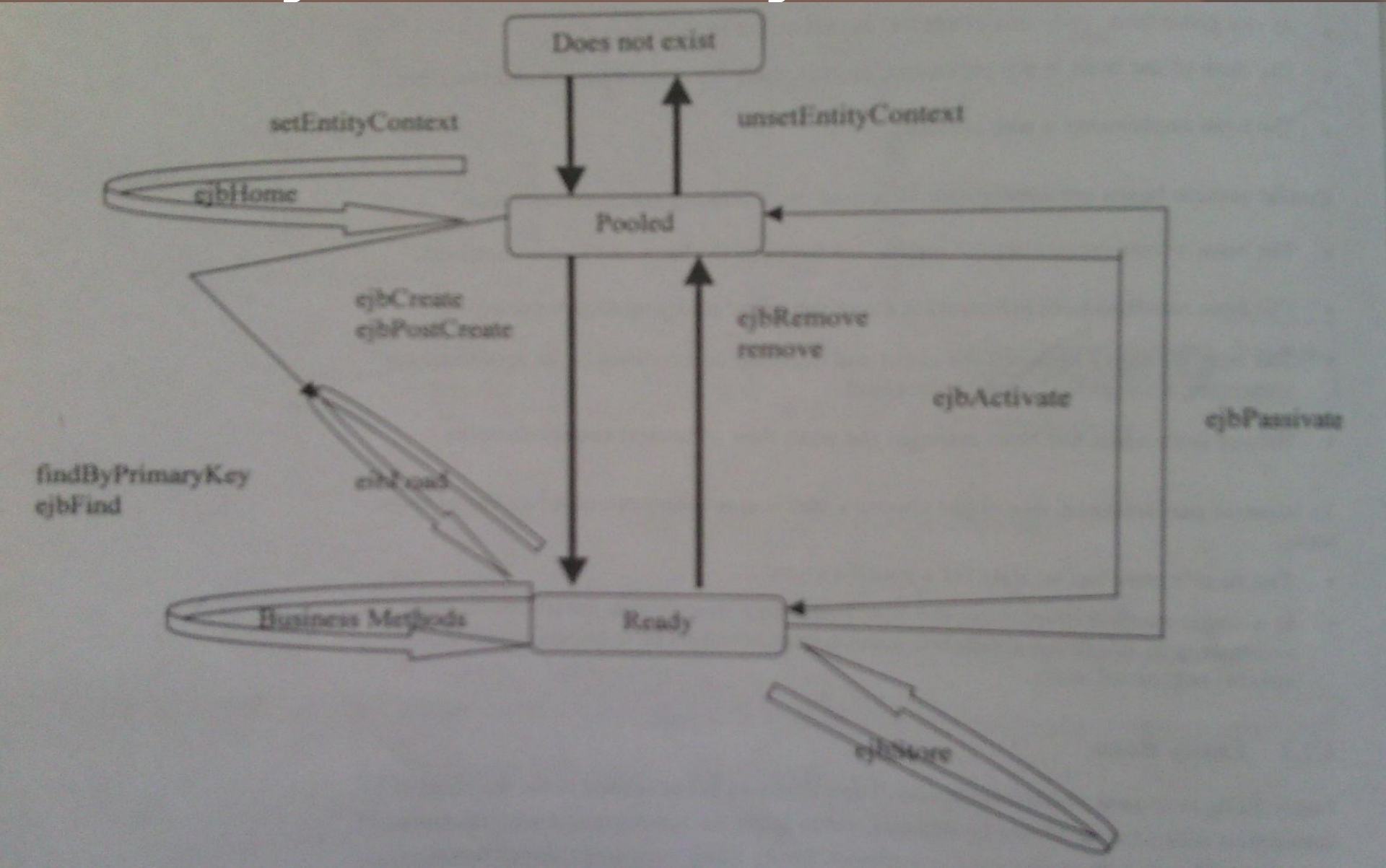
Continue...

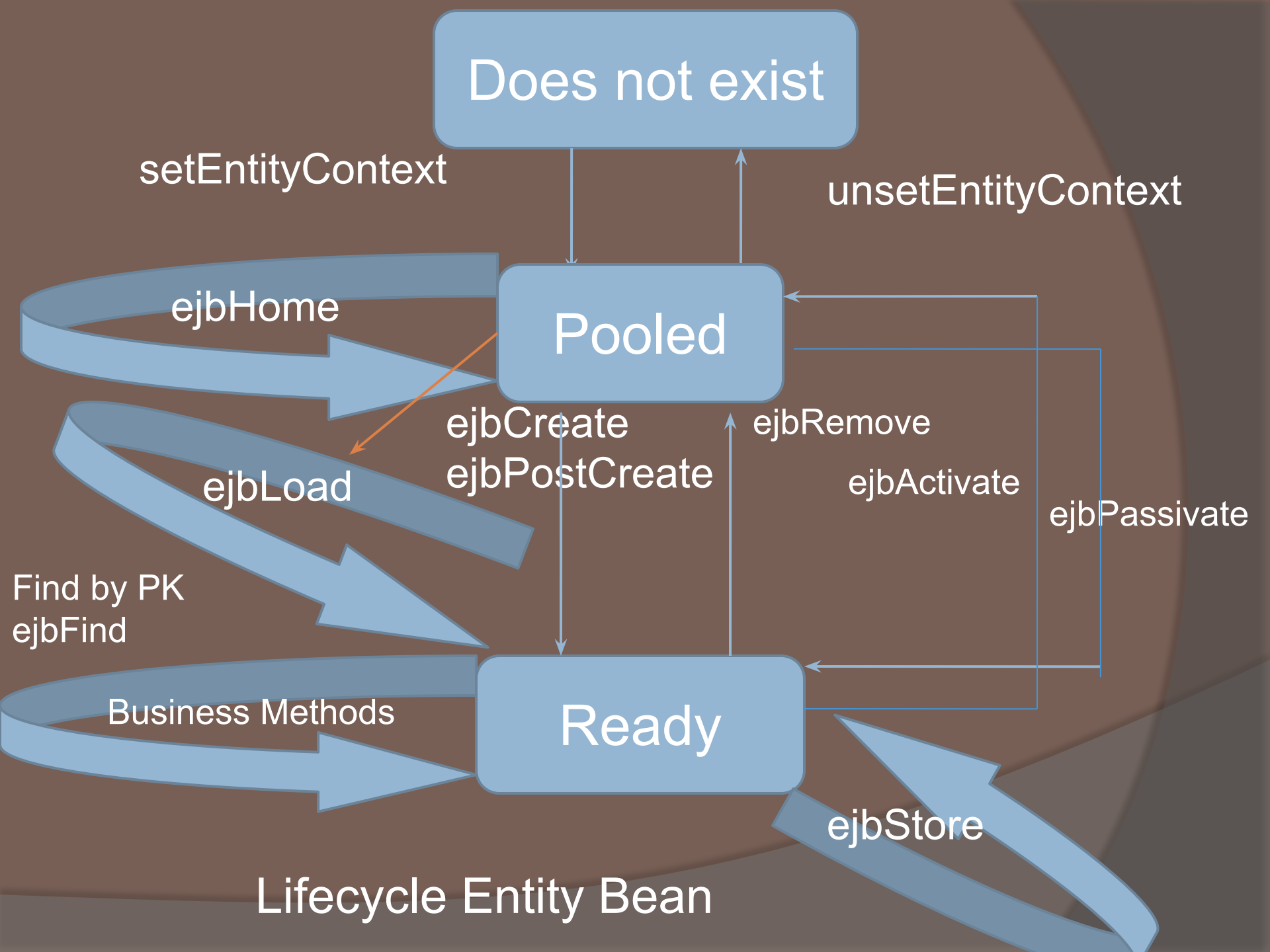
- An important feature of entity EJBs is that they have identity – that is, one can be distinguished from another.
- This is implemented by assigning a primary key to each instance of the EJB, where ‘primary key’ has the same meaning as it does for database management.
- Primary keys that identity EJBs can be of any type, including programmer – defined classes.
- There are two type of persistence that entity EJB supports.

Continue….

- These persistence types are:
- Bean – managed persistence (BMP):
  - The entity bean's implementation manages persistence by coding database access and updating statements in callback methods.
- Container – managed persistence (CMP):
  - The container uses specifications made in the deployment descriptor to perform database access and update statements automatically.

Lifecycle of Entity Bean





Lifecycle Entity Bean

Continue...

- Does not exist:
  - In this state, the bean instance simply does not exist.
- Pooled state:
  - When the EJB server is first started, several bean instances are created and placed in the pool. a bean instance in the pooled state is not tied to a particular data, that is, it does not correspond to a record in a database table.
  - Additional bean instances can be added to the pool as needed, and a maximum number of instances can be set.

Continue...

- Ready state:
  - A bean instance in the ready state is tied to a particular data, that is, it represents an instance of an actual business object.

Methods of Entity Bean

- setEntityContext:
  - This method is called for, if a container wants to increase its pool size of bean instances, then it will instantiate a new entity bean instance.
  - This method associates a bean with context information.
  - Once this method is called for, then, the bean can access the information about its environment.
- ejbFind(…)
  - This method is also known as the finder method. The finder method locates one or more existing entity bean data instances in underlying persistent store.

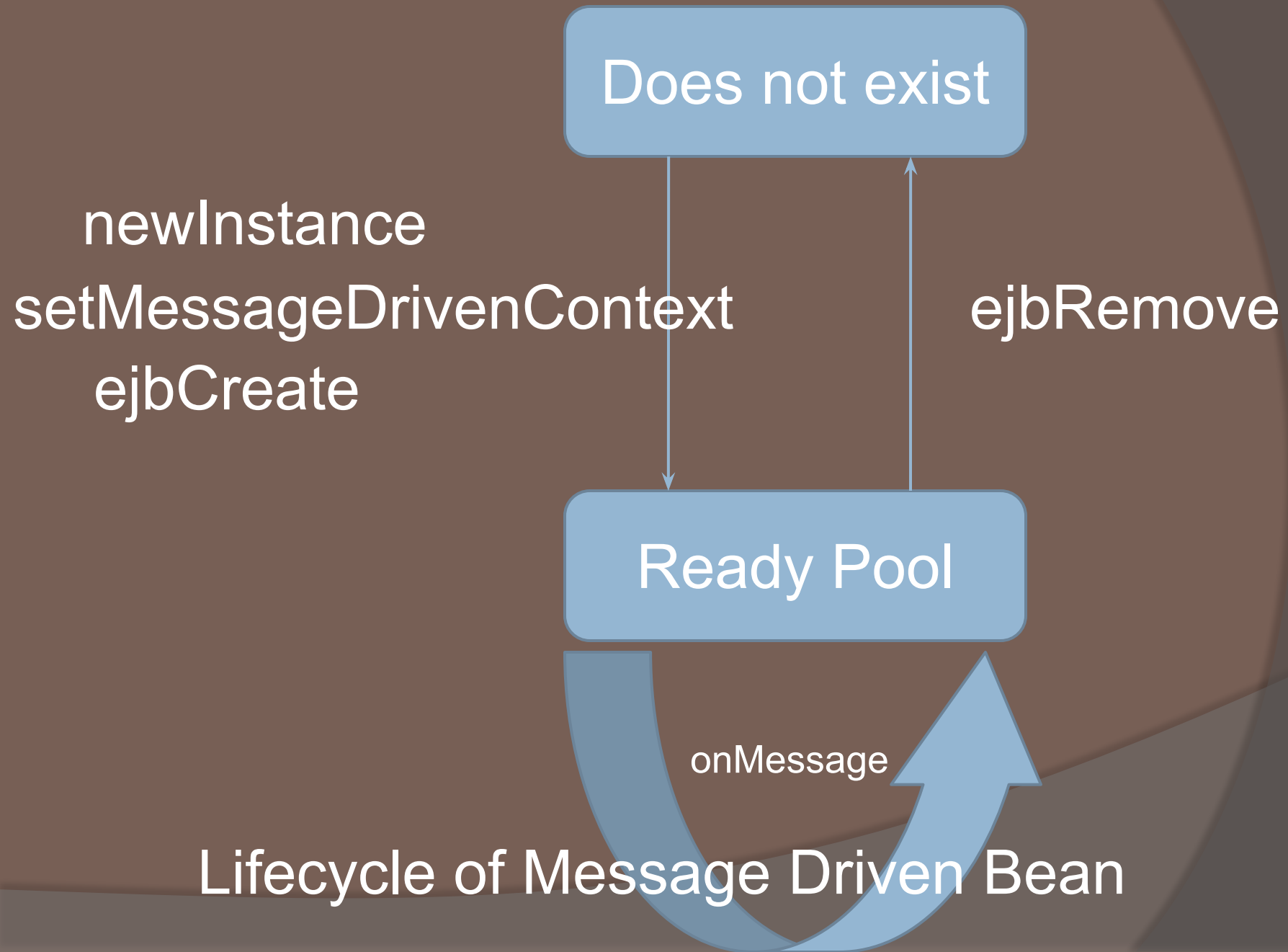
- ejbHome(…)
  - The home methods are special business because they are called from a bean in the pool before the bean is associated with any specific data. The client calls for, home methods from home interface or local home interface.
- ejbCreate()
  - This method is responsible for creating a new database data and for initialising the bean.
- ejbPostCreate()
  - There must be one ejbPostCreate() for each ejbCreate(). Each method must accept the same parameters. The container calls for, ejbPostCreate() right after ejbCreate().

- ejbActivate()
  - When a client calls for, a busiess method on a EJB object but no entity bean instance is bound to EJB object, the container needs to take a bean from the pool and transition into a ready state.
  - This is called activation. Upon activation the ejbActivate() is called for by the ejb container.
- ejbLoad()
  - This mehod is called for, to load the database in the bean instance.
- ejbStore()
  - This method is used for, to update the database with new values from the memory. This method is also called for during ejbPassivate().

- ejbPassivate()
  - This method is called for, by the EJB container when an entity bean is moved from the ready state to the pool state.
- ejbRemove()
  - This method is used to destroy the database data. It does not remove the object. The object is moved to the pool state for reuse.
- unsetEntityContext()
  - This method removes the bean from its environment. This is called for, just before destroying the entity bean.

Message Driven Bean

- A message – driven bean acts as a consumer of asynchronous messages. It cannot be called for, directly by clients, but is activated by the container when a message arrives.
- Clients interact with these EJBs by sending messages to the queues or topics to which they are listening.
- Although a message – driven EJB cannot be called for, directly by clients, it can call other EJBs itself.



Lifecycle of Message Driven Bean

- Does not exist:
  - In this state, the bean instance simply does not exist. Initially, the bean exists in the; does not exist state.
- Pooled state:
  - After invoking the ejbCreate(), the MDB instance is in the ready pool, waiting to consume incoming messages.
  - Since, MDBs are stateless, all instances of MDBs in the pool are identical; they're allocated to process a message and then return to the pool.

Methods

- onMessage(Message):
  - This method is invoked for each message that is consumed by the bean. The container is responsible for serializing messages to a single message driven bean.
- ejbCreate():
  - When this method is invoked, the MDB if first created and then, added to the ‘pool’.
- ejbRemove():
  - When this method is invoked, the MDB is removed from the ‘pool’.

Continue...

- setMessageDrivenContext(MessageDrivenContext):
  - This method is called for, as a part of the event transition that message driven bean goes through, when it is being added to the pool. This is called for, just before the ejbCreate().

Restrictions On EJB

- Enterprise Java Beans have several restrictions, which they must adhere to:
- They cannot create or manage threads,
- They cannot access threads using the java.io package.
- The cannot operate directly with sockets.
- They cannot load native libraries.
- They cannot use the AWT to interact with the user.
- They can only pass objects and values which are compatible with RMI.
- They must supply a public no arg. Constructor.
- Methods cannot be static or final.
- There are more minor restrictions.

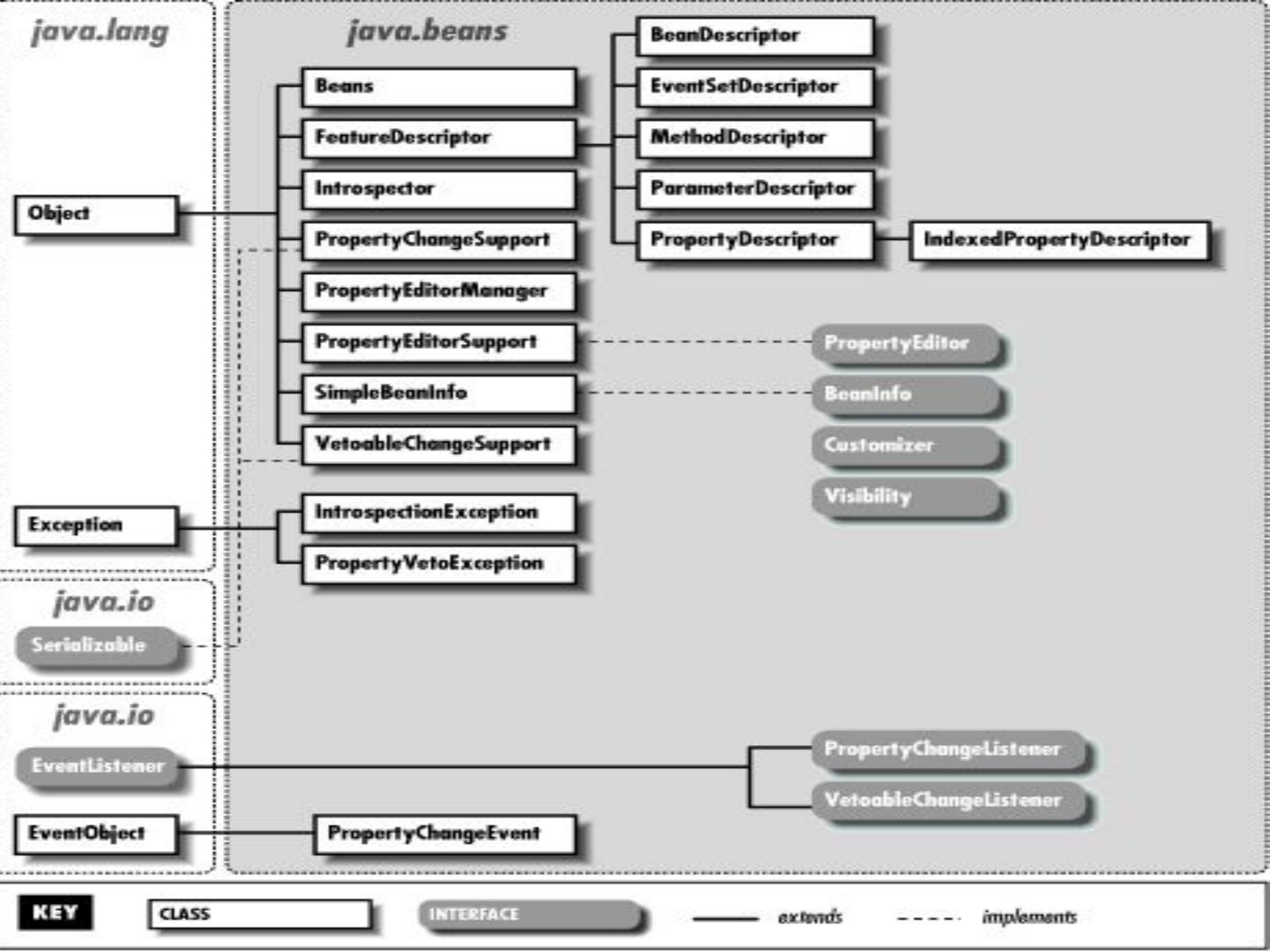
- Following can be done with EJB:
- Subclass another class.
- Interfaces can extend other interfaces, which are descendants of EJBObject or EJBHome.
- Helper methods can pass any kind of parameters or return types within the EJB.
- Helper methods can be any kind of visibility.

Timer Service

- Timer service is a mechanism using which scheduled application can be build. For example, salary slip generation on 1st of every month. EJB 3.0 specification has specified @Timeout annotation which helps in programming the ejb service in a stateless or message driven bean.
- EJB Container calls the method which is annotated by @Timeout.
- EJB Timer Service is a service provided by Ejb container which helps to create timer and to schedule callback when timer expires.

Java Bean Package

- Java.beans:
  - The classes and packages in the java.beans package can be categorized into three subtypes:
  - Design Support:
    - Classes – Beans, PropertyEditorManager, PropertyEditorSupport
    - Interfaces – Visibility, VisibilityState, PropertyEditor, Customizer
  - Introspection Support:
    - Classes – Introspector, SimepleBeanInfo, BeanDescriptor, EventSetDescriptor, etc.
    - Interfaces - BeanInfo


java.lang
java.beans
BeanDescriptor
Beans
EventSetDescriptor
FeatureDescriptor
MethodDescriptor
Introspector
ParameterDescriptor
Object
PropertyChangeSupport
PropertyDescriptor
IndexedPropertyDescriptor
PropertyEditorManager
PropertyEditorSupport
PropertyEditor
SimpleBeanInfo
BeanInfo
VetoableChangeSupport
Customizer
Visibility
Exception
IntrospectionException
PropertyVetoException
java.io
Serializable
java.io
EventListener
PropertyChangeListener
VetoableChangeListener
EventObject
PropertyChangeEvent
KEY
CLASS
INTERFACE
extends
implements