Managing Entities
Java Persistance API (JPA)

Contents

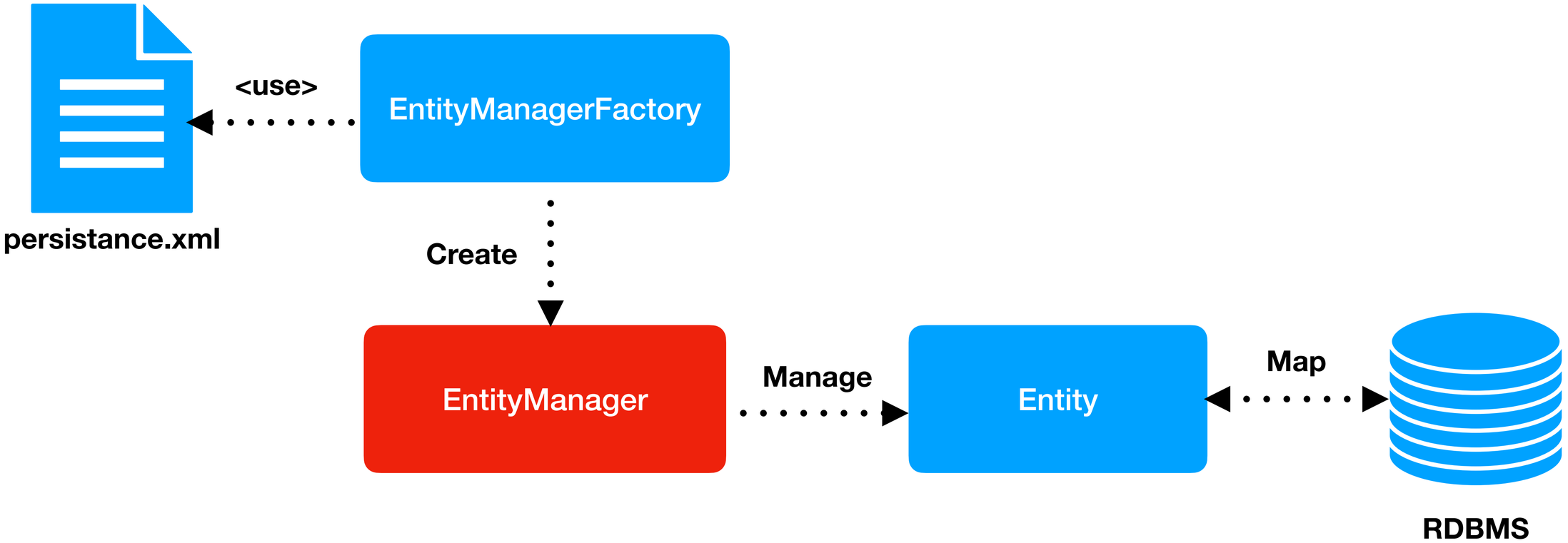
Hibernate
JPA

- Entity Manager
- Persistance Unit
- Manipulating Entities
- Cascade Operations
- Persistence Context
- Lifecycle Callbacks & Listeners

Entity Manager

- JPA ရဲ့ အဓိက Interface တစ်ခုဖြစ်ပြီး Entity Object တစ်ခုရဲ့ Lifecycle ကို Manage လုပ်ပေးနိုင်ပါတယ်
- Entity လေးတစ်ခုရဲ့ Lifecycle Changes တွေအလိုက် Database Operation တွေကို လုပ်ဆောင်ပေးမှာ ဖြစ်ပါတယ်
- EntityManager Object ရဲ့ နောက်ကွယ်မှာ PersistanceContext ဆိုတဲ့ Object လေးတစ်ခုဟာ ပူးတွဲတည်ရှိနေတတ်ပါတယ်
- EntityManager Object ရဲ့ သက်တမ်းတစ်လျောက်မှာ PersistenceContext တစ်ခုဟာ တည်ရှိနေပြီး၊ EntityManager ကနေ Manage လုပ်ထားတဲ့ Entity Object တွေကိုသိမ်းပေးထားတဲ့ Cache အနေနဲ့ အသုံးပြုပါတယ်
- PersistenceContext ဟာ Set ကို အခြေခံထားတဲ့ အတွက် သိမ်းထားတဲ့ Entity တွေဟာ Unique ဖြစ်နေစေပါတယ်

EntityManager



Creating EntityManager

- EntityManager တွေကို တည်ဆောက်ဖို့အတွက် EntityManagerFactory ကို အသုံးပြုပါတယ်။
- EntityManagerFactory ကို တည်ဆောက်တဲ့အခါမှာ JPA ရဲ့ Configuration ဖြစ်တဲ့ persistence.xml ကို အသုံးပြုပြီး တည်ဆောက်ရပါတယ်
- JPA Configuration ထဲမှာပါတဲ့ PersistenceUnit မှာသတ်မှတ်ထားတဲ့ အတိုင်း EntityManagerFactory ကို တည်ဆောက်ပေးနိုင်ပါတယ်
- Application တစ်ခုမှာ EntityManagerFactory ကို တစ်ကြိမ်သာ တည်ဆောက်ရပြီး၊ EntityManager ကိုတော့ လိုအပ်သလို အကြိမ်ကြိမ် တည်ဆောက်ယူနိုင်ပါတယ်
- EntityManager ကိုတည်ဆောက်တဲ့အခါမှာ Java SE ပတ်ဝန်းကျင်နဲ့ Java EE ပတ်ဝန်းကျင်တို့မှာ ရေးသားပုံချင်း မတူကြပါဘူး

Java SE Environment

```java
public class ItemDao {

    private EntityManager em;

    public ItemDao() {
        EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("zhygo");
        this.em = emf.createEntityManager();
    }

    public void create(Item item) {
        this.em.getTransaction().begin();
        this.em.persist(item);
        this.em.getTransaction().commit();
    }

}
```

Java EE Environment

```java
@Transactional
public class DataManager<T> {

    @PersistenceContext(name="zhygo-web")
    protected EntityManager em;

    @Loggable
    public void persist(T t) {
        this.em.persist(t);
    }

    public T selectById(Object obj, Class<T> clz) {
        return this.em.find(clz, obj);
    }
}
```

Persistence Unit

- JPA Project တွေမှာ Deployment Descriptor အနေနဲ့ persistence.xml File တစ်ခုကို ရေးသားရန်လိုအပ်ပြီး၊ အသုံးပြုမည့် Database ပတ်ဝန်းကျင်နဲ့ ပတ်သက်ပြီး Persistence Unit ကို သတ်မှတ်ရေးသားရပါတယ်

- EntityManagerFactory ကိုတည်ဆောက်တဲ့ အခါမှာ Persistence Unit ထဲမှာ သတ်မှတ်ထားတဲ့ Configuration ကိုအသုံးပြုပြီး တည်ဆောက်သွားမှာ ဖြစ်ပါတယ်

- EntityManager တွေကိုတည်ဆောက်တဲ့ အခါမှာ ရေးသားထားတဲ့ Configuration တွေကို ပြန်ပြီး အသုံးပြုသွားမှာ ဖြစ်ပါတယ်

Standard Options for Persistence Unit

Hibernate

JPA

- Datasource Configurations
- Schema Generation
- Database Initializations
- Validations
- Others Configurations

Datasource Configuration

Configurations	Values
javax.persistence.transactionType	JTA, RESOURCE_LOCAL
javax.persistence.jtaDataSource	JTA Datasource name
javax.persistence.nonJtaDataSource	Non JTA Datasource name
javax.persistence.jdbc.driver	JDBC Driver class name for Java SE
javax.persistence.jdbc.url	JDBC URL for Java SE
javax.persistence.jdbc.user	Database user name for Java SE
javax.persistence.jdbc.password	Database password for user

Schema Generation

Configurations	Values
database.action	none, create, drop-and-create, drop
script.action	none, create, drop-and-create, drop
create-source	metadata, script, metadata-then-script, script-then-metadata
drop-source	metadata, script, metadata-then-script, script-then-metadata
create-database-schemas	true, false

Schema Generation - Script

Configurations	Values
scripts.create-target	output of create ddl (sql) file name
scripts.drop-target	output of drop ddl (sql) file name
scripts.create-script-soruce	url of DDL Script file
scripts.drop-script-source	url of DDL Script file
javax.persistence.sql-load-script-source	SQL load script for database initialization

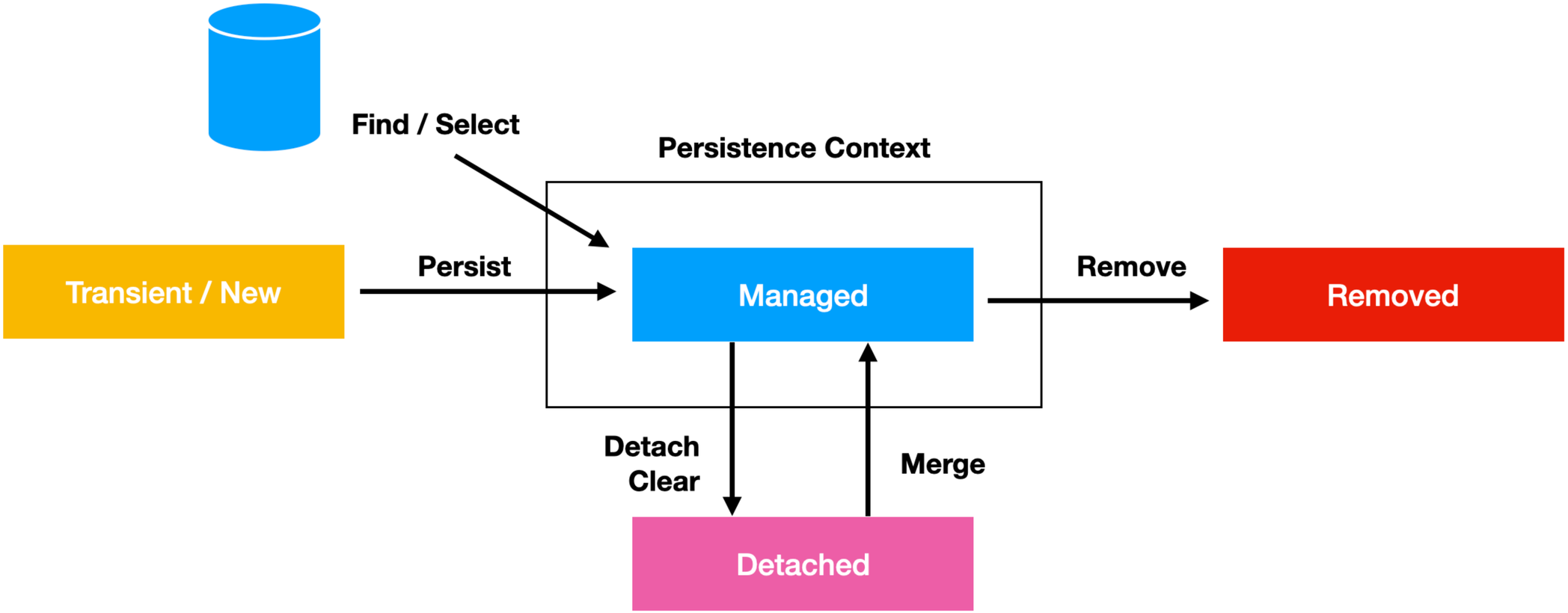
Validations & Others

Configurations	Values
javax.persistence.validation.group.pre-persist	Group of Validation Classes which want to execute pre persist state
javax.persistence.validation.group.pre-update	Group of Validation Classes which want to execute pre update state
javax.persistence.validation.group.pre-remove	Group of Validation Classes which want to execute pre remove state
javax.persistence.lock.timeout	
javax.persistence.query.timeout	

Manipulating Entity

Return Type	Method
void	persist (Object entity)
<T> T	find(Class<T> entityClass, Object id)
<T> T	getReference(Class<T> entityClass, Object id)
void	detach(Object entity)
void	clear()
<T> T	merge(T entity)
void	remove (Object entity)
void	referesh (Object entity)
void	flush ()
boolean	contains(Object entity)

States of Entity

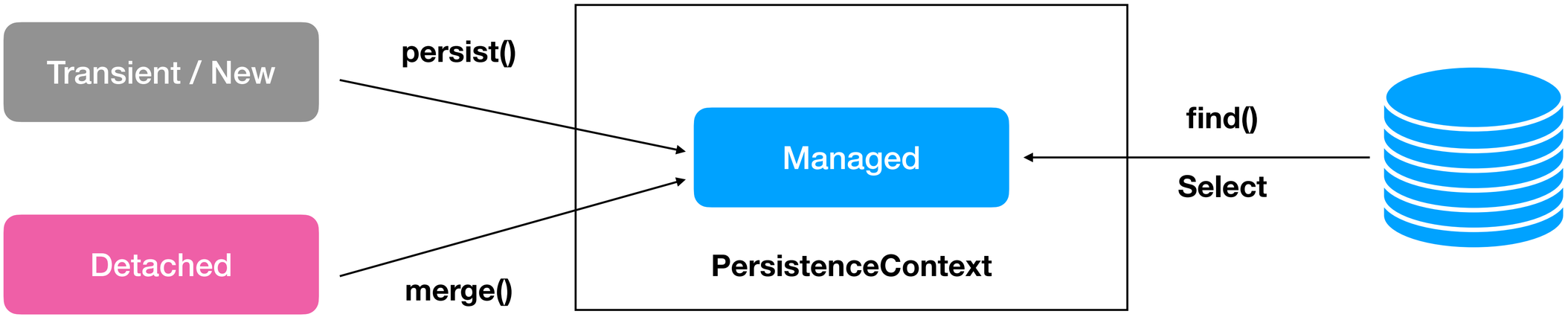


Transient / New State

- Entity Object လေးတစ်ခုကို တည်ဆောက်ပြီးကာစ၊ EntityManager ကနေ Manage မလုပ်ရသေးတဲ့ အနေအထား

- ရိုးရိုး Java Object လေးအနေနဲ့ပဲ တည်ရှိနေပြီး Database နဲ့ ပတ်သက်မှုရှိမှာ မဟုတ်သေးပါဘူး

Managed State



- EntityManager ကနေ Manage လုပ်ထားတဲ့အနေအထားဖြစ်ပြီး၊ Database နှင့် Synchronous ဖြစ်နေပါတယ်
- Managed State ဖြစ်တဲ့ Entity Object လေးရဲ့ Fields တွေကို Setter တွေကနေ ပြောင်းလဲပေးတိုင်း Transaction ကို Commit လုပ်တဲ့ အခါမှာ နောက်ဆုံးအနေအထားနဲ့ Database ကို Update လုပ်ပေးနိုင်မှာ ဖြစ်ပါတယ်

Persisting Entity

- Transient State မှာရှိတဲ့ Entity Object လေးကို EntityManager ရဲ့ persist() Method ဖြင့် persist လုပ်လိုက်ရင် Managed State ကို ရောက်ရှိသွားမှာ ဖြစ်ပြီး Transaction ကို Commit လုပ်တဲ့ အခါ Database ထဲကို Insert လုပ်ပေးသွားပါမယ်

- Persist လုပ်မည့် Entity ကနေ Reference လုပ်နေတဲ့ အခြား Entity တွေရှိနေရင်၊ Reference လုပ်နေတဲ့ Entity တွေရဲ့ State ဟာ Managed State ဖြစ်နေမှ Persist လုပ်လို့ရမှာ ဖြစ်ပါတယ်

- Persist လုပ်မည့် Entity ကနေ Reference လုပ်နေတဲ့ အခြား Entity က Transient State ဖြစ်နေရင်တော့ TransientPropertyValueException ကို ဖြစ်စေမှာ ဖြစ်ပါတယ်

Find By ID

- EntityManager မှာ ID ကို အသုံးပြုပြီး Entity တွေကို ရှာဖွေနိုင်ဖို့ Method နှစ်မျိုးရှိပါတယ်

- ID နဲ့ရှာလာတဲ့ Entity Object တွေဟာလဲ EntityManager ကနေ Manage လုပ်ထားပြီး Managed State ကို ရောက်ရှိနေမှာ ဖြစ်ပါတယ်

Feature	Find Method	Get Reference Method
Select By	Primary Key	Primary Key
Fetch Members	According to Fields Fetch Modes	Lazy Fetch Modes for all Fields
If not Found	null	EntityNotFoundException

Fetch Modes

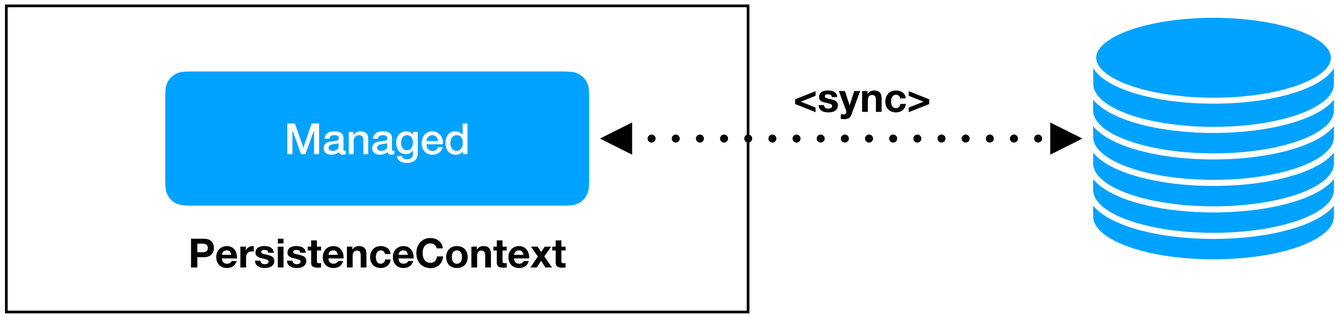
- Entity ရဲ့ Attribute တွေမှာ Fetch Mode တွေရှိကြပြီး၊ သတ်မှတ်ထားခြင်းမရှိရင်တောင်မှ Data Type အလိုက် Default Fetch Mode တွေရှိကြပြီး၊ လိုအပ်ပါက Default Fetch Mode တွေကို ပြောင်းလဲ သတ်မှတ်ပေးနိုင်ပါတယ်

- Eager Mode ဆိုပါက Database ထဲကနေ Select လုပ်ကထဲက Projection Field တွေထဲမှာတစ်ခါထဲ Select လုပ်လာခဲ့မှာ ဖြစ်ပြီး၊ Lazy Mode ဆိုပါက Select လုပ်တဲ့ အခါမှာ Projection Field ထဲမှာပါမှာမဟုတ်ပါဘူး

- Lazy Mode Attribute တွေကို Managed State ထဲမှာရှိတဲ့ အချိန် getter method တွေကနေ Access လုပ်မှသာ Database ထဲကနေ Select လုပ်ပေးမှာ ဖြစ်ပါတယ်

- တကယ်လို့ Lazy Mode Attribute တွေကို Access လုပ်တဲ့အခါမှာ Session ရှိနေတဲ့ အနေအထားမှာပဲ Fetch လုပ်လို့ရမှာဖြစ်ပြီး၊ EntityManager ကို Close လုပ်လိုက်ပြီးတဲ့အခါမှ Access လုပ်မိရင်တော့ LazyInitializationException ကို ဖြစ်ပေါ်စေမှာ ဖြစ်ပါတယ်

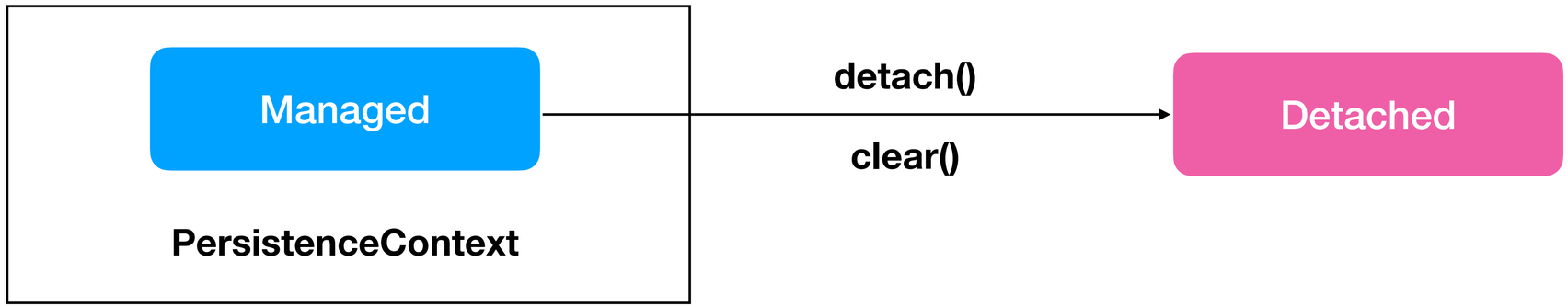
Update fields in Managed State



- Entity တစ်ခုဟာ Managed State မှာရှိနေတယ် ဆိုတာဟာ Database နဲ့ Synchronize ဖြစ်နေတဲ့ အနေအထားဖြစ်ပါတယ်

- Managed State မှာရှိတဲ့ Entity Object တစ်ခုရဲ့ Fields တွေကို Setter Method တွေကနေ Update လုပ်ထားရင် နောက်ဆုံး Transaction ကို Commit လုပ်တဲ့ အခါမှာ SQL Update Statement ကို အသုံးပြုပြီး Update လုပ်ပေးမှာ ဖြစ်ပါတယ်

Detached State



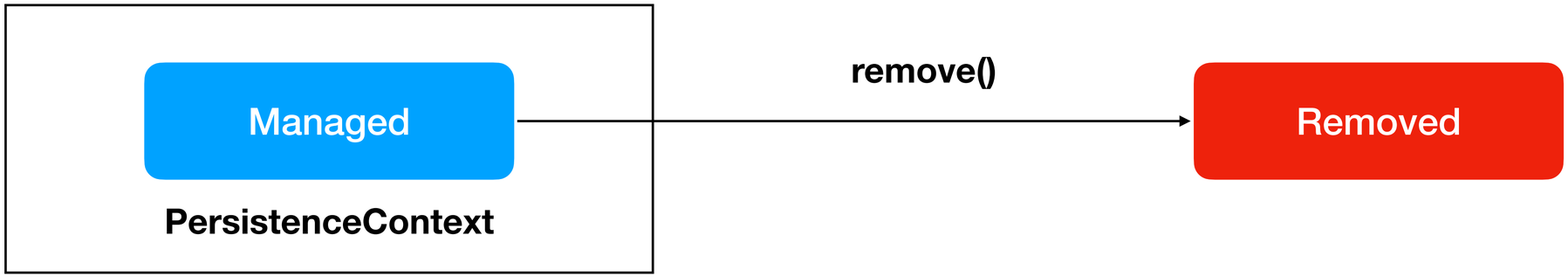
- EntityManager ကနေ Manage လုပ်မထားတော့တဲ့ အနေအထားဖြစ်ပါတယ်
- EntityManager ရဲ့ detach() Method ကို အသုံးပြုပြီး Managed State မှာရှိတဲ့ Entity ကို Detached State ကို ရောက်အောင်ပြုလုပ်နိုင်သလို၊ clear() Method ကို အသုံးပြုပြီး Managed လုပ်ထားတဲ့ Entity အားလုံးကို Detached State ကို ရောက်ရှိအောင် ပြုလုပ်နိုင်ပါတယ်
- Detached State ကို ရောက်သွားရင် Entity ကို PersistenceContext ထဲကနေ ထုတ်ပစ်လိုက်ပြီး Database နဲ့ Synchronize လုပ်မပေးတော့ပါဘူး

Merging Entity

- Detached State ကို ရောက်ရှိသွားတဲ့ Entity Object တစ်ခုကို Managed State ကို ရောက်အောင်ဆိုရင် EntityManager ရဲ့ merge() method ကို အသုံးပြုနိုင်ပါတယ်

- Detached State ဖြစ်နေတုန်း Setter Method နဲ့ Entity ရဲ့ Fields တွေကို ပြင်ထားတာရှိရင်လဲ Transaction ကို Commit လုပ်တဲ့ အခါမှာ Update Statement နဲ့ Database ကို Update လုပ်ပေးမှာ ဖြစ်ပါတယ်

Removed State



- Managed State မှာရှိတဲ့ Entity Object လေးကို EntityManager ရဲ့ remove() Method နဲ့ Removed State ကို ရောက်ရှိအောင် ဆောင်ရွက်နိုင်ပါတယ်

- Removed State ဆိုတာကတော့ Database ထဲကနေ Delete လုပ်မည့် Entity တွေရဲ့ State ဖြစ်ပြီး Transaction ကို Commit လုပ်တဲ့ အခါ Database ထဲကနေ Delete လုပ်သွားပါမယ်

Orphan Removable

- Entity Object တစ်ခုဟာ အခြားသော Entity တစ်ခုရဲ့ Private Member အဖြစ်ပါဝင်နေပြီး အဲ့ဒီ Entity သာ မရှိတော့ရင်၊ သူ့ကို Reference လုပ်မဲ့ Object လဲ ရှိမှာမဟုတ်တော့ပါဘူး

- အဲ့ဒီအခါမျိုးမှာ Persistence Provider ကနေ အလိုအလျောက်ဖျက်စီးပေးနိုင်အောင် စီမံထားနိုင်ပါတယ်

- one-to-one နဲ့ one-to-many association တွေမှာ orphan-removable ကို support လုပ်ထားပြီး ၎င်းရဲ့ တန်ဖိုးကို true လုပ်ထားပါက Owner Object ကို remove လုပ်တဲ့အခါမှာ အလို အလျောက် child object ကို လည်း Remove လုပ်သွားမှာ ဖြစ်ပါတယ်

State Changes for Persist Operation

Original State	After Persist	Description
New / Transient	Managed	Insert Operation ကိုလုပ်ဆောင်ပြီး Managed State ကို ရောက်သွားပါမယ်
Managed	Managed	ပြောင်းလဲမှုရှိမှာမဟုတ်ပါဘူး
Detached	PersistenceException	Detach State မှာရှိတဲ့ Entity Object ကို Persist လုပ်လို့ မရနိုင်ပါဘူး
Removed	Managed	Delete လုပ်ထားတဲ့ Object ကို ပြန်ပြီး Insert လုပ်ပြီး Managed State ကို ရောက်ရှိစေပါမယ်

State Changes for Merge Operation

Original State	After Merge	Description
New / Transient	Managed	Insert Operation ကိုလုပ်ဆောင်ပြီး Managed State ကို ရောက်သွားပါမယ်
Managed	Managed	ပြောင်းလဲမှုရှိမှာမဟုတ်ပါဘူး
Detached	Managed	Detached ဖြစ်နေတဲ့ Entity ကို Setter ကနေ Update လုပ်ထားရင် Managed State Object ကို Commit လုပ်တဲ့ အခါ Database ကို
Removed	IllegalArgumentException	Removed State မှာရှိတဲ့ Entity တွေကို Merge လုပ်လို့ မရနိုင်ပါဘူး

State Changes for Remove Operation

Original State	After Remove	Description
New / Transient	Nothing	ဘာမှဖြစ်မှာ မဟုတ်ပါဘူး
Managed	Removed	နောက်ဆုံးမှာတော့ Database ကနေ Delete လုပ်သွားပါမယ်
Detached	IllegalArgumentException	Detach State မှာရှိတဲ့ Entity ကို Remove လုပ်လို့ မရနိုင်ပါဘူး
Removed	Nothing	နောက်ဆုံးမှာတော့ Database ကနေ Delete လုပ်သွားပါမယ်

Checking State of Entity

- Detached State မှာရှိတဲ့ Entity Object ကို persist() လုပ်တဲ့အခါနဲ့ remove() လုပ်တဲ့အခါတွေမှာ Exception တွေကို ဖြစ်ပေါ်စေနိုင်ပါတယ်

- Persist ဒါမှမဟုတ် Remove မလုပ်ခင် Entity Object ကို Manage State မှာရှိရဲ့လား ဆိုတာကို EntityManager ရဲ့ contains() Method နဲ့ စစ်ဆေးနိုင်ပါတယ်

- Entity Object လေးဟာ Manage State မှာရှိနေမယ် ဆိုရင် contains() Method ရဲ့ Result ဟာ true ဖြစ်ပြီး မဟုတ်ရင် false ဖြစ်နေမှာ ဖြစ်ပါတယ်

Cascade Operations

- JPA Entity တွေကို ရေးသားတဲ့ အခါမှာ Entity တစ်ခုထဲမှာ အခြားသော Entity တွေကို Relationship တွေကို သတ်မှတ်ပြီး ရေးသားထားလေ့ရှိပါတယ်
- EntityManager တွေနဲ့ Entity Object တွေကို Manipulate လုပ်တဲ့ အခါမှာ Default အတိုင်းဆိုရင် Target Entity ကိုပဲ လုပ်ဆောင်ပေးတာဖြစ်ပါတယ်
- ဒါပေမဲ့ တစ်ခါတစ်လေမှာ Entity Object တစ်ခုကို Operation လုပ်လိုက်တဲ့အခါမှာ Relationship ရှိတဲ့ တစ်ခြား Entity တွေကို အလားတူ Operation မျိုး ဆောင်ရွက်စေလိုတဲ့ အခါလဲရှိနိုင်ပါတယ်
- အဲ့ဒီလိုလုပ်ပေးနိုင်အောင် JPA မှာ Cascade Operation တွေကို ဆောင်ရွက်နိုင်အောင် ပြင်ဆင်ပေးထားပါတယ်
- Cascade Operation တွေကို Relationship Mapping တွေမှာ သတ်မှတ်ရေးသားပေးနိုင်ပါတယ်

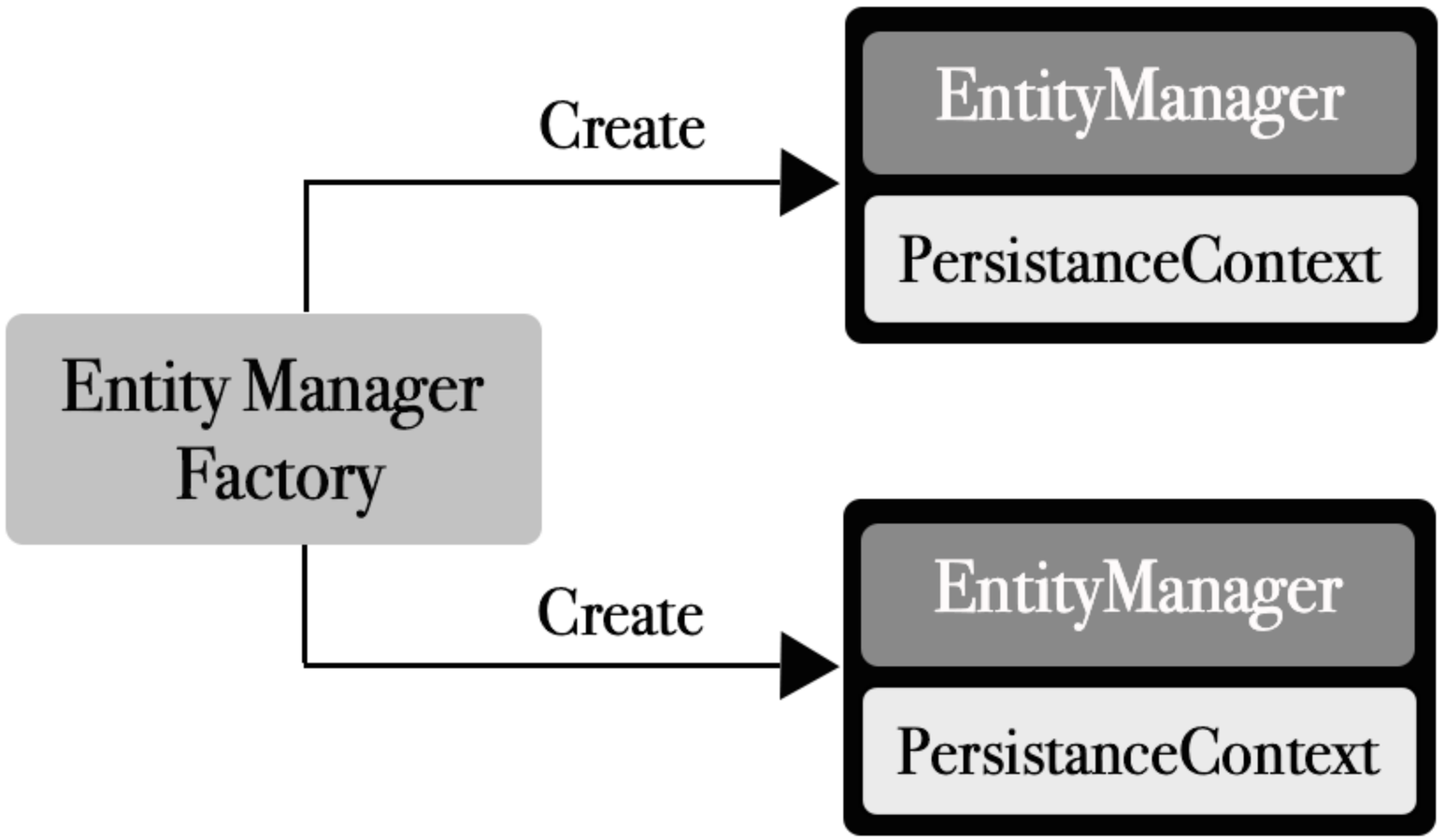
Cascade Types

- Relationship Annotation တွေမှာ cascade attribute မှာ CascadeType [] ကို သတ်မှတ်ရေးသားနိုင်ပါတယ်
- PERSIST, MERGE, REMOVE, DETACH, REFRESH တန်ဖိုးတွေကို လိုအပ်သလိုသတ်မှတ် အသုံးပြုနိုင်မှာ ဖြစ်ပါတယ်
- တဖန် ALL ကိုလဲ ပြင်ဆင်ပေးထားပြီး၊ Relationship Owner Entity ရဲ့ Operation တိုင်းကို Reference Entity ကို ဆောင်ရွက်ပေးသွားမှာ ဖြစ်တဲ့အတွက် သတိပြုပြီး အသုံးပြုသင့်ပါတယ်
- Cascade Mode တွေကို Relationship Annotation ရှိတဲ့ နေရာတိုင်းမှာ ရေးသားအသုံးပြုနိုင်ပြီး၊ Relationship Owner ဘက်မှာကော Mapped By ဘက်မှာပါ Cascade Operation တွေကို ရေးသားအသုံးပြုနိုင်ပါတယ်

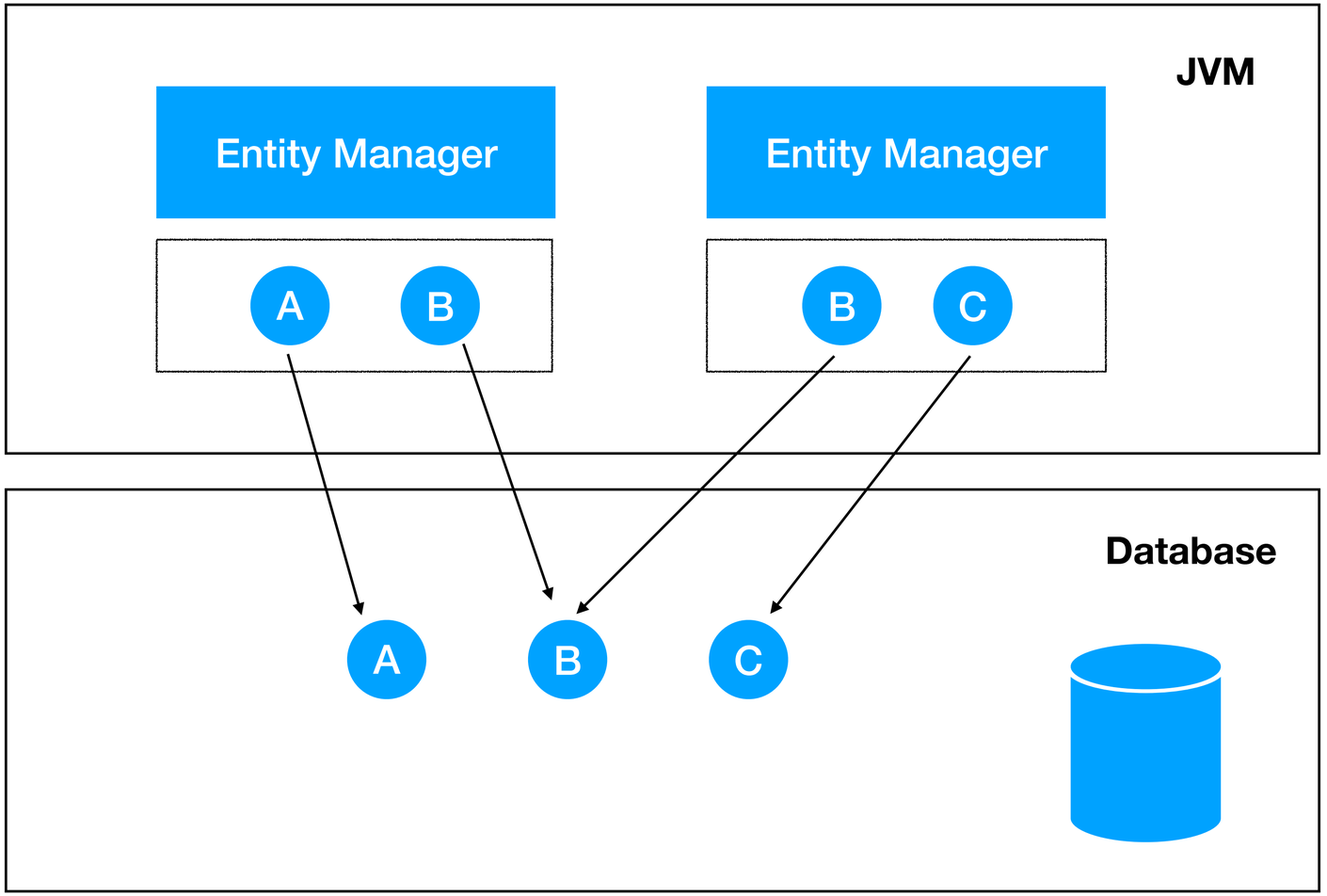
Persistence Context

- EntityManager တစ်ခုကို Create လုပ်ပြီဆိုတာနဲ့ ၎င်းနှင့်အတူ PersistenceContext ကိုပါတည်ဆောက်ပေးလိုက်ပါတယ်
- EntityManager တည်ရှိနေသမျှ PersistenceContext ဟာလဲ အတူတည်ရှိနေပါမယ်
- EntityManager ရဲ့ method invocation တွေကြောင့် Managed State ကိုရောက်ရှိနေတဲ့ Entity Object တွေဟာ PersistenceContext ထဲမှာတည်ရှိနေပါမယ်
- PersistenceContext တွေဟာ SET Structure ကို အခြေခံထားတဲ့ အတွက် Managed လုပ်ထားတဲ့ Entity Object တွေကို Duplication ဖြစ်နေလို့မရပါဘူး
- EntityManager ကို Close လုပ်လိုက်ပြီဆိုတာနဲ့ PersistenceContext ကိုလဲ ရှင်းပစ်မှာ ဖြစ်တဲ့အတွက် Entity Object တွေဟာ Detached State ကိုရောက်ရှိသွားမှာ ဖြစ်ပါတယ်

Persistance Context



Persistence Context



Synchronizing with DB

- JPA ၏ EntityManager ဟာ First Level Cash ဖြစ်ပြီး၊ ပုံမှန်အားဖြင့် Transaction ကို Commit လုပ်လိုက်သော အခါမှ Manage လုပ်ထားသော Entity များကို Database အတွင်းသို့ Flash လုပ်လိုက်မည်ဖြစ်သည်

- သို့ရာတွင် EntityManager ၏ flush နှင့် refresh method တို့ကို အသုံးပြုပြီး မိမိလိုအပ်သော အခါတွင် Database နှင့် Synchronize လုပ်နိုင်ပါသည်

- flush method ကို ခေါ်ဆိုလိုက်ပါက EntityManager သည် ပြုပြင်ပြောင်လည်းမှုများကို Database အတွင်းသို့ Flush ပြုလုပ်မည် ဖြစ်ပါသည်။ persist လုပ်ထားသော Object ကို flush လုပ်ပါက Commit မလုပ်မှီ၊ flush လုပ်သည့် အချိန်တွင် Database အတွင်းသို့ insert လုပ်မည် ဖြစ်ပါသည်

- တဖန် refresh method ကို ခေါ်ယူလိုက်ပါက Database အတွင်းမှတန်ဖိုးဖြင့် Entity Object အားပြန်လည် ပြောင်းလည်းသွားမည် ဖြစ်သည်

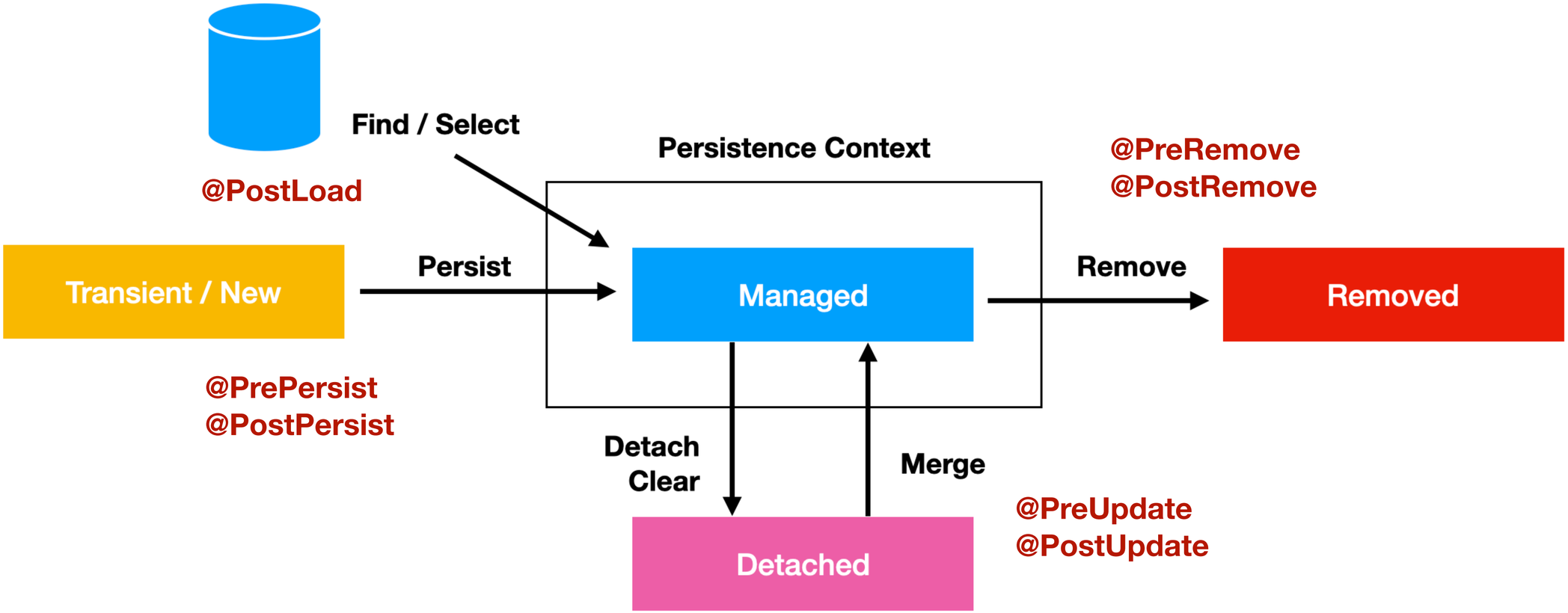
Synchronization Methods

Method	Return Type & Description
flush()	PersistenceContext ထဲက State တွေနဲ့ Database ကို Synchronize လုပ်ပေးနိုင်ပါတယ်
refresh(Object)	Database ထဲက State တွေနဲ့ Entity Object လေးကို Refresh လုပ်ပေးနိုင်မှာ ဖြစ်ပါတယ်
refresh(Object, LockModeType)	
refresh(Object, LockModeType, Map<String,Object>)	
refresh(Object, Map<String,Object>)	

Lifecycle Callbacks & Listeners

- JPA မှာ Entity တစ်ခုရဲ့ State တွေပြောင်းလဲခြင်းအားဖြင့် Database Operation တွေကို ဆောင်ရွက်ပေးပါတယ်
- Entity လေးရဲ့ Lifecycle State တွေပြောင်းလဲမှု Event တွေဖြစ်တဲ့အခါတွေမှာ လုပ်ဆောင်မှုတွေကို ဆောင်ရွက်နိုင်ရန် Callback Methods တွေနဲ့ Listener Class တွေကို ရေးသားနိုင်အောင် ပြင်ဆင်ထားပါတယ်
- Entity တစ်ခုချင်းစီမှာလဲ Callback Method တွေကို ရေးသားနိုင်သလို၊ Callback Method တွေကို Class တစ်ခုထဲမှာ စုစည်းပြီး Listener Class အနေနဲ့လဲ ရေးသားအသုံးပြုနိုင်ပါတယ်
- Listener တွေကိုလဲ အသုံးပြုလိုတဲ့ Entity တွေမှာ သတ်မှတ်ရေးသားနိုင်သလို၊ Global Listener အနေနဲ့လဲ သတ်မှတ်ရေးသားနိုင်ပါတယ်

Lifecycle Callback



Lifecycle Hooks Annotations

Annotation	Description
@PrePersist	Entity ကို Persist မလုပ်ခင်မှာ လုပ်ဆောင်စေလိုတဲ့ Method တွေမှာ ရေးသားနိုင်ပါတယ်
@PostPersist	Entity ကို Persist လုပ်ပြီးတဲ့အခါ လုပ်ဆောင်စေလိုတဲ့ Method တွေမှာ ရေးသားနိုင်ပါတယ်
@PostLoad	Database ထဲကနေ Select လုပ်ပြီးတဲ့အခါမှာ လုပ်ဆောင်စေလိုတဲ့ Method တွေမှာ ရေးသားနိုင်ပါတယ်
@PreUpdate	Entity ကို Update မလုပ်ခင်မှာ လုပ်ဆောင်စေလိုတဲ့ Method တွေမှာ ရေးသားနိုင်ပါတယ်
@PostUpdate	Entity ကို Update လုပ်ပြီးတဲ့အခါ လုပ်ဆောင်စေလိုတဲ့ Method တွေမှာ ရေးသားနိုင်ပါတယ်
@PreRemove	Entity ကို Remove မလုပ်ခင်မှာ လုပ်ဆောင်စေလိုတဲ့ Method တွေမှာ ရေးသားနိုင်ပါတယ်
@PostRemove	Entity ကို Remove လုပ်ပြီးတဲ့အခါ လုပ်ဆောင်စေလိုတဲ့ Method တွေမှာ ရေးသားနိုင်ပါတယ်

Lifecycle Hooks

Lifecycle Hooks	Description
Callback Methods	Entity Class တွေထဲမှာ Lifecycle Hooks Annotation များကို အသုံးပြုပြီး Callback Method တွေကို ရေးသားနိုင်ပါတယ်
Listener Class	Lifecycle Callback Method တွေကို Class တစ်ခုထဲမှာ စုစည်းပြီး Entity Class တွေရဲ့ Lifecycle Listener Class အနေနဲ့ သတ်မှတ်ရေးသားနိုင်ပါတယ်
Global Listener	Lifecycle Listener Class တွေကို OR Mapping Configuration File တွေမှာ Global Listener အနေနဲ့ သတ်မှတ်ရေးသားနိုင်ပါတယ်

Lifecycle Callbacks

```java
@Entity
public class Account {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private int balance;

    private LocalDateTime creation;
    private LocalDateTime modification;

    @PrePersist
    public void beforeCreate() {
        creation = LocalDateTime.now();
        System.out.println("Create Account");
    }

    @PreUpdate
    public void beforeUpdate() {
        modification = LocalDateTime.now();
        System.out.println("Update Account");
    }

    // Getters & Setters

}
```

Entity Class ထဲမှာ Lifecycle Hooks Annotation တွေကို အသုံးပြုပြီး Callback Method တွေကို ရေးသားနိုင်ပါတယ်

Listener Class

```java
public class TimesListener {

    @PrePersist
    public void beforeCreate(Object object) {
        if(object instanceof TimeEnableEntity entity) {
            var times = entity.getTimes();
            if(null == times) {
                times = new Times();
                entity.setTimes(times);
            }
            times.setCreation(LocalDateTime.now());
        }
    }

    @PreUpdate
    public void beforeUpdate(Object object) {
        if(object instanceof TimeEnableEntity entity) {
            var times = entity.getTimes();
            if(null == times) {
                times = new Times();
                entity.setTimes(times);
            }
            times.setModification(LocalDateTime.now());
        }
    }
}
```

Class တစ်ခုထဲမှာ Callback Method တွေကို စုစည်းရေးသားပြီး Listener Class အနေနဲ့ အသုံးပြုနိုင်ပါတယ်

Entity Listener

```java
@Entity
@EntityListeners(TimesListener.class)
public class Member implements TimeEnableEntity {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private String loginId;
    private String password;
    @Enumerated(EnumType.STRING)
    private Role role;

    private Times times;

    // Getters & Setters
}
```

Listener ကို အသုံးပြုလိုတဲ့ Entity မှာ
@EntityListeners Annotation ဖြင့်
သတ်မှတ် အသုံးပြုနိုင်ပါတယ်

Default Listeners

```xml
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm http://xmlns.jcp.org/xml/ns/persistence/orm_2_2.xsd">
    <persistence-unit-metadata>
        <persistence-unit-defaults>
            <entity-listeners>
                <entity-listener
                    class="com.jdc.demo.entity.listeners.TimesListener"></entity-listener>
            </entity-listeners>
        </persistence-unit-defaults>
    </persistence-unit-metadata>
</entity-mappings>
```

- ရေးသားထားတဲ့ Listener တွေကို Default Listener အဖြစ် OR Mapping File ထဲမှာ သတ်မှတ်ရေးသားနိုင်ပါတယ်
- Default Listener တွေကို မသုံးစေလိုတဲ့ Entity မှာတော့ @ExcludeDefaultListeners ကို အသုံးပြုပြီး Exclude လုပ်နိုင်ပါတယ်