Simplified DAO Pattern with Generics

```
<<interface>>
GenericDao<T>
```

bind Bier

```
GenericDaoJpa<T>
```
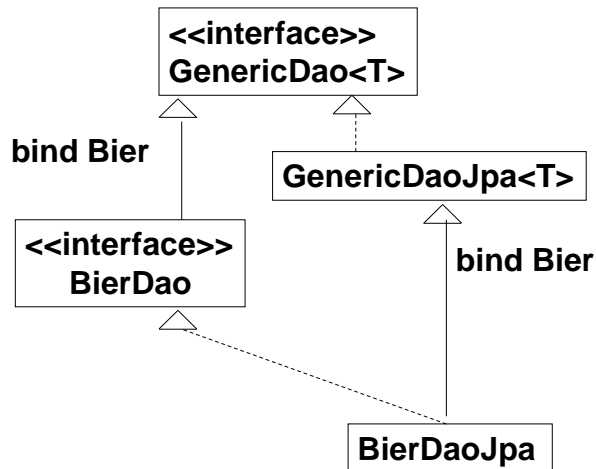
```
<<interface>>
BierDao
```

bind Bier

```
BierDaoJpa
```

Simplified DAO Pattern with Generics

```java
package repository;
import java.util.List;

public interface GenericDao<T> {

   public List<T> findAll();
   public T get(Long id);
   public T update(T object);
   public void delete(T object);
   public void insert(T object);
   public boolean exists(Long id);

}
```

Simplified DAO Pattern with Generics

```java
package repository;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class GenericDaoJpa<T> implements GenericDao<T> {

  private static final String PU_NAME = "bierWinkels";
  private static final EntityManagerFactory emf =
      Persistence.createEntityManagerFactory(PU_NAME);
      protected static final EntityManager em =
                          emf.createEntityManager();
  private final Class<T> type;

  public GenericDaoJpa(Class<T> type) {
    this.type = type;
  }
```

3

GenericDaoJpa<T>

```java
  public static void closePersistency() {
    em.close();
    emf.close();
  }

  public static void startTransaction() {
    em.getTransaction().begin();
  }

  public static void commitTransaction() {
    em.getTransaction().commit();
  }

  public static void rollbackTransaction() {
    em.getTransaction().rollback();
  }
```

4

```java
@Override
public List<T> findAll() {
    return em.createQuery("select entity from " +
        type.getName() + " entity", type).getResultList();
    /*return em.createNamedQuery(type.getName()+
        ".findAll", type).getResultList();*/
}

@Override
public T get(Long id) {
    T entity = em.find(type, id);
    return entity;
}

@Override
public T update(T object) {
    return em.merge(object);
}
```

5

GenericDaoJpa<T>

```java
@Override
public void delete(T object) {
    em.remove(em.merge(object));
}

@Override
public void insert(T object) {
    em.persist(object);
}

@Override
public boolean exists(Long id) {
    T entity = em.find(type, id);
    return entity != null;
}

}
```

6

Simplified DAO Pattern with Generics

```
package repository;

import domain.Bier;
import javax.persistence.EntityNotFoundException;

public interface BierDao extends GenericDao<Bier>  {

    public Bier getBierByName(String name)
                throws EntityNotFoundException;
}
```

Simplified DAO Pattern with Generics

```
package repository;

import domain.Bier;
import javax.persistence.EntityNotFoundException;
import javax.persistence.NoResultException;

public class BierDaoJpa extends GenericDaoJpa<Bier>
                        implements BierDao {

  public BierDaoJpa() {
     super(Bier.class);
  }
```

Simplified DAO Pattern with Generics

```java
@Override
public Bier getBierByName(String name)
        throws EntityNotFoundException {
  try {
    return em.createNamedQuery("Bier.findByName", Bier.class)
        .setParameter("bierNaam", name)
        .getSingleResult();
  } catch (NoResultException ex) {
    throw new EntityNotFoundException();}
  }
}
```

```java
@Entity
@NamedQueries({
    @NamedQuery(name = "Bier.findByName",
                  query = "select b from Bier b where b.naam = :bierNaam")
})
public class Bier implements Serializable {
```

9

Use DAO

```java
public class DomeinController {

    private GenericDao<Winkel> winkelRepo;
    private BierDao  bierRepo;

…

  //IN CONSTRUCTOR:
  setWinkelRepo(new GenericDaoJpa<>(Winkel.class));
  setBierRepo(new BierDaoJpa());

…

  //SETTER-INJECTIONS
  public void setWinkelRepo(GenericDao<Winkel> mock){
      winkelRepo = mock;
  }
  public void setBierRepo(BierDao mock){
      bierRepo = mock;
  }
```

10

5

Use DAO

```
...
public void voegBierBijWinkel(String bierNaam, String winkelNaam)
                        throws IllegalArgumentException {
    Optional<Winkel> winkel = getWinkelList().stream()
        .filter( w -> w.getNaam().equalsIgnoreCase(winkelNaam))
        .findFirst();
    if (!winkel.isPresent()) {
        throw new IllegalArgumentException("winkel " +
                                winkelNaam + " komt niet voor");
    }
    Bier bier;
    try {
        bier = bierRepo.getBierByName(bierNaam);
    } catch(EntityNotFoundException ex) {
        throw new IllegalArgumentException("bier " + bierNaam +
                                " komt niet voor");
    }
    GenericDaoJpa.startTransaction();
    winkel.get().addBier(bier);
    GenericDaoJpa.commitTransaction();
}
```

11

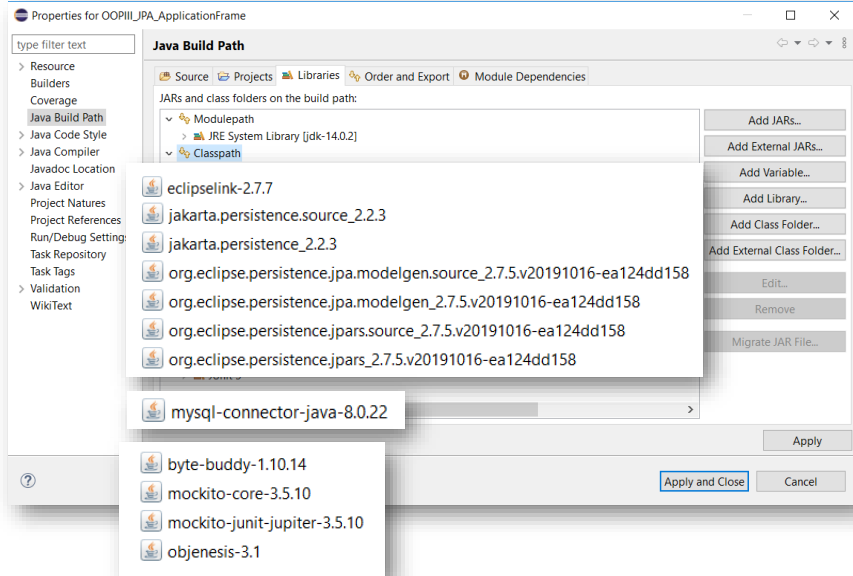## Use DAO

```
...
    private List<Winkel> getWinkelList(){
        if (winkelList==null){
            winkelList=winkelRepo.findAll();
        }
        return winkelList;
    }

...                         [Op den hoek, Putteke]


    public void close() {
        GenericDaoJpa.closePersistency();
    }
...
```
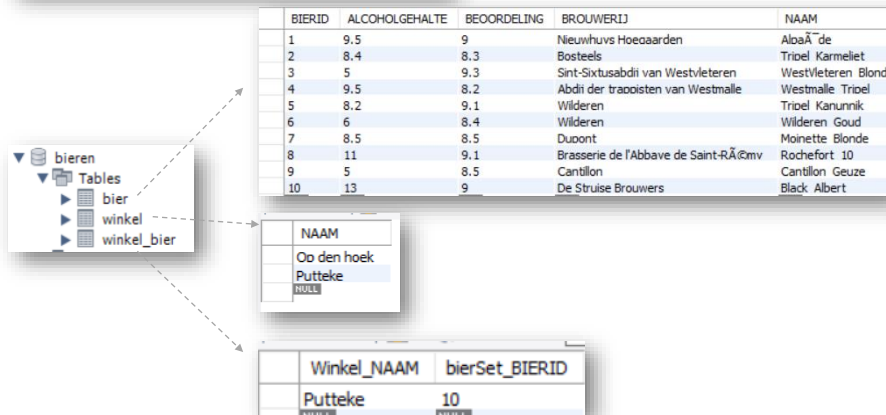
12

13

14

Testklasse

```java
@ExtendWith(MockitoExtension.class)
public class DomeinTest {

    @Mock
    private GenericDao<Winkel> winkelRepo;

    @Mock
    private BierDao bierRepo;

    @InjectMocks
    private DomeinController domein;

    @Test
    public void voegBierBijWinkel() {
```

Runs: 1/1	Errors: 0	Failures: 0	

DomeinTest [Runner: JUnit 5] (1,377 s)
　voegBierBijWinkel() (1,377 s)

≡ Failure Trace