

# Software Engineering: Specification, Implementation, Verification

## Chapter 5: Data Management

Suad Alagić

*Springer 2017*

# Main topics

- Implementing use cases
- Persistence
- File - based persistence
- Transactions
- Object – relational technology
- Representing associations
- Relational representation
- Representing inheritance
- Queries

- Uses cases as transactions
- Object lifetimes and persistence
- Managing persistent objects
- Database technologies
- Transaction technologies

# Implementing use cases

**Use case:** BuyAsset

**Entities:** Investor, Asset, Portfolio, Broker

**Actors:** Investor, Broker

**Constraints:**

**Preconditions:** assetPriceOK, brokerApproves

**PostCondition:** assetInPortfolio

**Frame:** All other assets in portfolio unaffected

# Implementing use cases

**Use case:** SellAsset

**Entities:** Investor, Asset, Portfolio, Broker

**Actors** Investor, Broker

**Constraints:**

**Preconditions:** assetInPortfolio, brokerApproves

**Postcondition:** not assetInPortfolio

**Frame:** All other assets in portfolio unaffected

# Transactions

- *Atomicity*
- *Consistency*
- *Isolation*
- *Durability*

- *Orthogonality*
- *Transitivity (reachability)*
- *Transparency*

```
public class Aircraft
{ private String model;
  private Pilot pilot;
  public Aircraft(String aModel)
  { model = aModel; pilot = null; }
  public assignPilot(Pilot p)
  { pilot=p;}
  // other methods
}
```



```
public class Pilot
{ private String name;
  private int points;
  public Pilot(String pName, int pPoints)
  { name = pName; points = pPoints; }
  // other methods
}
```

# Persistence

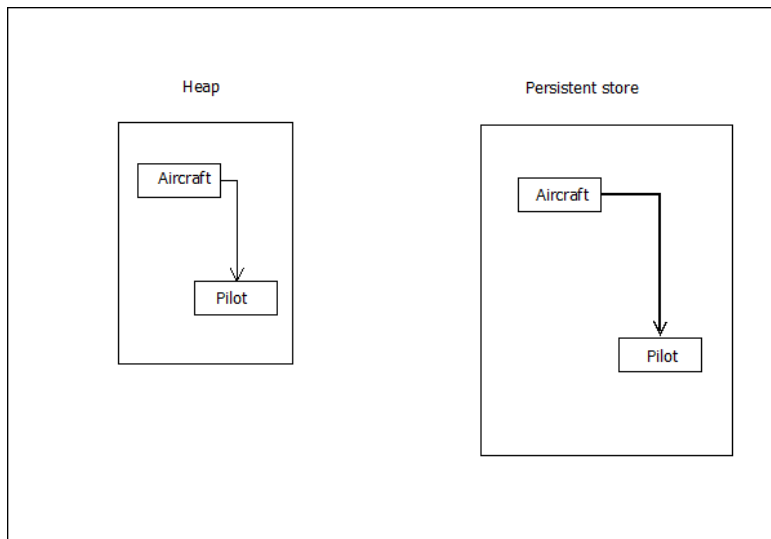


Figure: Persistent complex objects

# File based persistence

- Java model of persistence
- Serialized objects
- Orthogonality
- Transitivity
- Transparency

# File based persistence

```
public interface ObjectOutputStream extends DataOutput
{
    void writeObject(Object obj);
        throws IOException;
    // other methods
}
```

```
public interface ObjectInputStream extends DataInput
{
    Object readObject()
        throws ClassNotFoundException, IOException;
    // other methods
}
```

# File based persistence

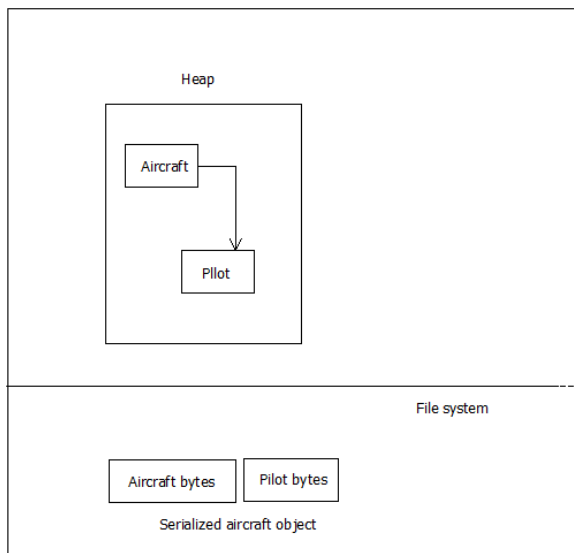


Figure: Serialized complex object

# File based persistence

```
public class Aircraft implements Serializable {  
    // . . .  
}  
FileOutputStream fileOut = new FileOutputStream("AircraftFile");  
ObjectOutput out = new ObjectOutputStream(fileOut);  
aircraftObj = new Aircraft("Boeing777");  
out.writeObject(aircraftObj);  
out.flush();  
out.close();  
  
FileInputStream fileIn = new FileInputStream("AircraftFile");  
ObjectInput in = new ObjectInputStream(fileIn);  
Aircraft aircraftObj= (Aircraft) in.readObject();  
in.close();
```

- JDO model of persistence
- Persistent capable classes
- Persistence manager
- Class extents
- Transaction begin, start and rollback

```
@PersistenceCapable  
public class Aircraft  
{ . . .  
{
```

```
PersistentManagerFactory pmf = get persistent manager factory;  
PersistentManager pm = pmf.getPersistentManager();
```

```
pm.makePersistent(a)
```



# Transactions

```
Transaction Tx = pm.currentTransaction();  
try  
{ Tx.begin();  
  // transaction code  
  Tx.commit();  
}  
catch (Exception ex)  
{ exception handling }  
finally  
{   if (Tx.isActive())  
    Tx.rollback();  
}  
pm.close();  
}
```

# Transactions

```
Extent e = pm.getExtent(Aircraft.class)
```

```
Transaction Tx = pm.currentTransaction();
```

```
try
```

```
{ Tx.begin();
```

```
  Aircraft a= new Aircraft("Boeing 777");
```

```
  pm.makePersistent(a);
```

```
  Tx.commit();
```

```
}
```

```
Extent e = pm.getExtent(Aircraft.class)
```

```
catch (Exception ex)
{ exception handling }
finally
{
    if (Tx.isActive())
        Tx.rollback();
}
pm.close();
}
```

# Transactions

```
Transaction Tx = pm.currentTransaction();  
try  
{  
    Tx.begin();  
    Extent e = pm.getExtent(Aircraft.class);  
    Iterator it = e.iterator();  
    while (it.hasNext());  
    { Aircraft a = (Aircraft)it.next();  
      a.pilot.display();  
    }  
    Tx.commit();  
}
```

```
catch (Exception ex)
    { exception handling }
finally
{ if (Tx.isActive())
    Tx.rollback();
}
pm.close();
}
```

# Transactions

```
Transaction Tx = pm.currentTransaction();  
try  
{  
    Tx.begin();  
    Extent e = pm.getExtent(Aircraft.class);  
    Iterator it = e.iterator();  
    while (it.hasNext());  
    { Aircraft a = (Aircraft)it.next();  
      Pilot p = get pilot;  
      if (a.model = "Boeing 777")  
          a.assignPilot(p);  
    }  
    Tx.commit();  
}
```

```
catch (Exception ex)
    { exception handling }
finally
{ if (Tx.isActive())
    Tx.rollback();
}
pm.close();
}
```

# Object – relational technology

- Entity classes
- Persistent context
- Entity manager
- Transactions as resources
- Transparency



# Object – relational technology

```
@Entity  
public class Aircraft  
{ . . .  
}
```

```
@Entity  
public abstract class Aircraft  
{  
    @Id  
    protected String aircraftId;  
    . . .  
}
```

# Object – relational technology

@Entity

```
public class PassengerPlane extends Aircraft {  
    {  
        int capacity;  
        . . .  
    }
```

@Entity

```
public class CargoPlane extends Aircraft {  
    {  
        float maxLoad;  
        . . .  
    }
```

# Object – relational technology

```
@PersistentContext  
EntityManager em;
```

```
@Resource  
UserTransaction Tx;
```

```
@PersistentContext  
EntityManager em;  
public void findAircraft(String id)  
{  
    Aircraft a = em.find(Aircraft.class, id);  
    a.getPilot().display();  
}
```

# Object – relational technology

```
@PersistentContext
EntityManager em;
public void newAircraft(String id, String model)
{
    Aircraft a = new Aircraft("US1", "Boeing747");
    em.persist(a);
}
```

```
@PersistentContext
EntityManager em;
public void removeAircraft(String id)
{
    Aircraft a = em.find(id);
    em.remove(a);
}
```

# Representing associations

- Associations by methods
- One to one
- Many to one
- Many to many

# Representing associations

```
public class Investor
{ . . .
  @OneToOne
  public Portfolio getPortfolio() {
    return myPortfolio;
  }
}
```

```
public class Portfolio
{ . . .
  @OneToOne
  public Investor getOwner() {
    return owner;
  }
}
```

# Representing associations

```
public class Portfolio
{ . . .
  @ManyToOne
  public Broker getBroker() {
    return manager;
  }
```

```
public class Broker
{ . . .
  @OneToMany
  public Collection<Portfolio> getMyPortfolios() {
    // select portfolios of this broker;
  }
```

# Representing associations

```
public class Asset {  
    { . . .  
    @ManyToMany  
    public Collection<Portfolio> getMyPortfolios() {  
        // select portfolios having this assest;  
    }  
}
```

```
public class Portfolio  
{ . . .  
    @ManyToMany  
    public Collection<Asset> getPortfolioAssets() {  
        // select assets of this portfolio;  
    }  
}
```



# Relational representation

```
public class Investor {  
  @Id  
  String investorId;  
  String portfolioId; // foreign key  
  . . .  
}
```

```
public class Portfolio {  
  @Id  
  String portfolioId;  
  String ownerId; // foreign key  
  . . .  
}
```

# Relational representation

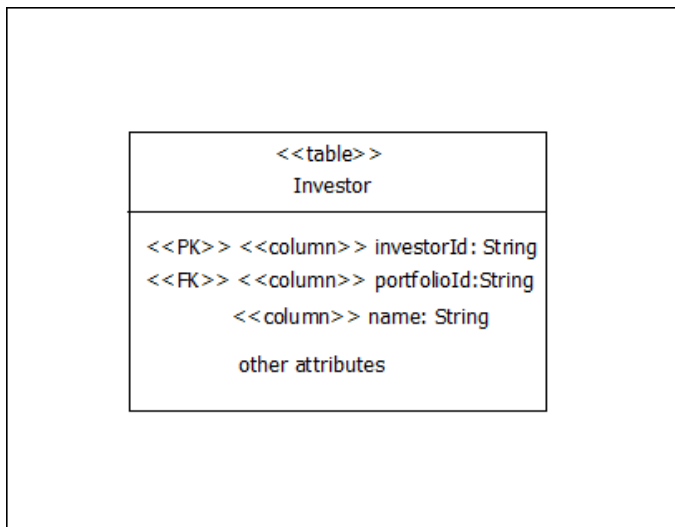


Figure: Investor table

# Relational representation

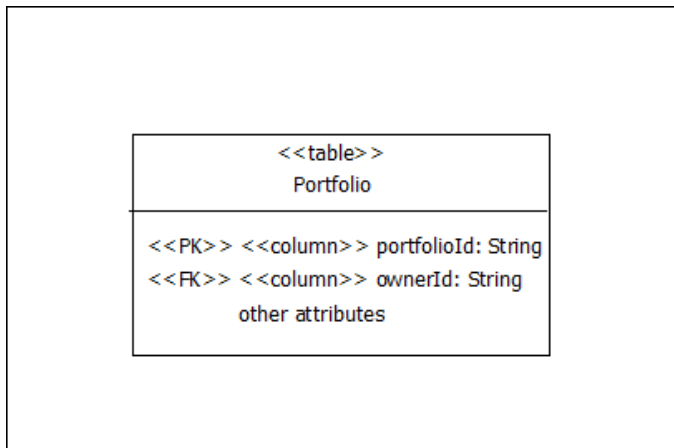


Figure: Portfolio table

# Relational representation

```
public class Broker {  
  @Id  
  String brokerId;  
  . . .  
}
```

# Relational representation

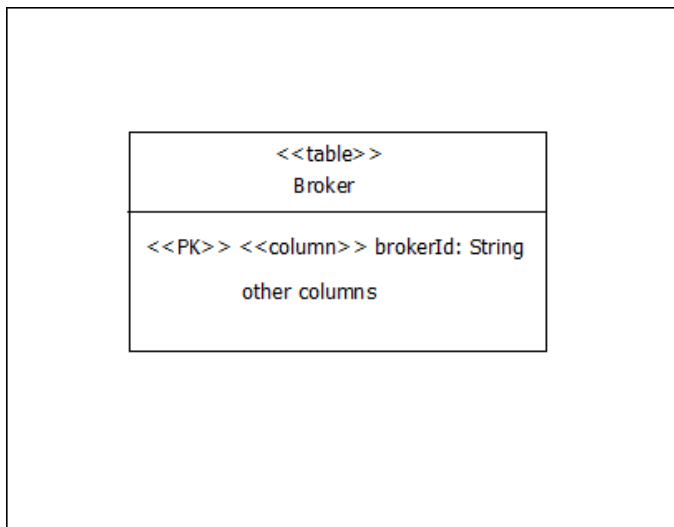


Figure: Broker table

# Relational representation

```
public class Portfolio {  
  @Id  
  String portfolioId;  
  String ownerId; // foreign key  
  String brokerId; // foreign key  
  . . .  
}
```

# Relational representation

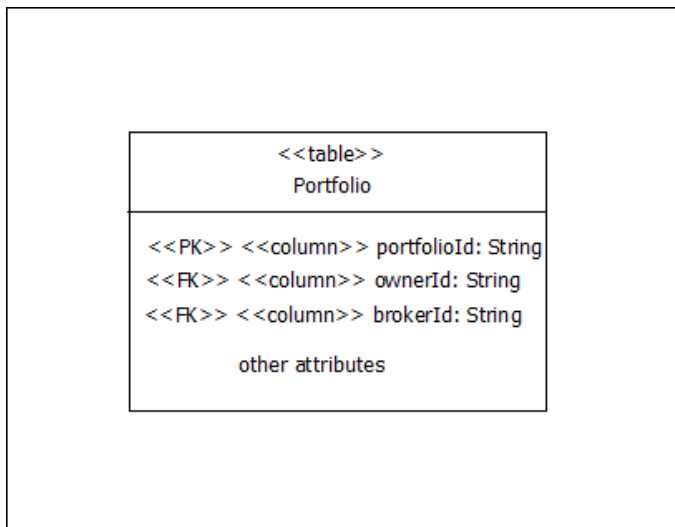


Figure: Portfolio table

# Relational representation

```
public class Asset {  
  @Id  
  String assetId;  
  . . .  
}
```



# Relational representation

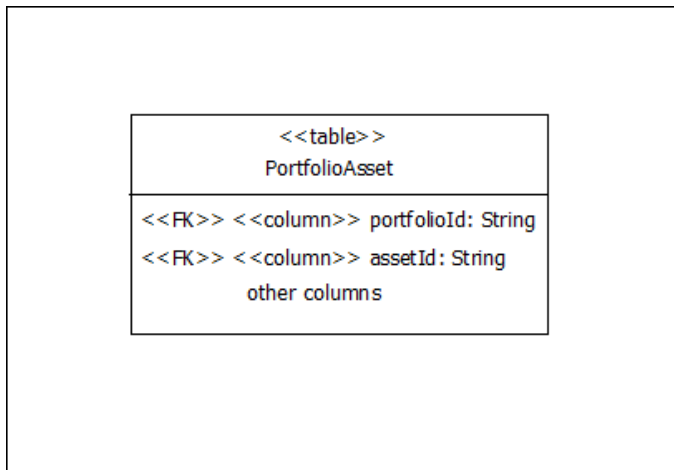


Figure: PortfolioAsset table

# Relational representation

- Primary and foreign keys
- UML notion of a table
- Associations by keys
- Representing inheritance

# Relational representation

```
public class PortfolioAsset {  
    String portfolioId; // foreign key  
    String assetId; // foreign key  
    . . .  
}
```

# Representing inheritance

```
public class Aircraft {  
    @Id  
    String aircraftId;  
    String model;  
    int capacity;  
    float maxLoad;  
    String discriminator;  
    . . .  
}
```

# Representing inheritance

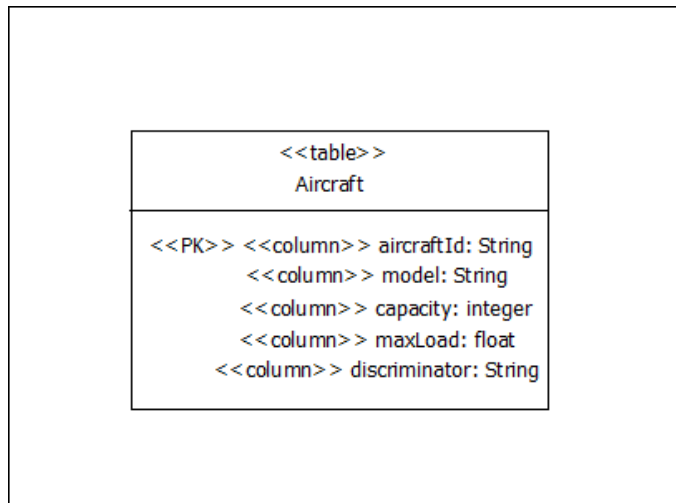


Figure: Aircraft table

# Representing inheritance

```
public class Aircraft {  
  @Id  
  String aircraftId;  
  String model;  
  . . .  
}
```

# Representing inheritance

```
public class PassengerPlane {  
    @Id  
    String aircraftId;  
    String model;  
    int capacity;  
    . . .  
}
```

```
public class CargoPlane {  
    @Id  
    String aircraftId;  
    String model;  
    float maxLoad;  
    . . .  
}
```

# Representing inheritance

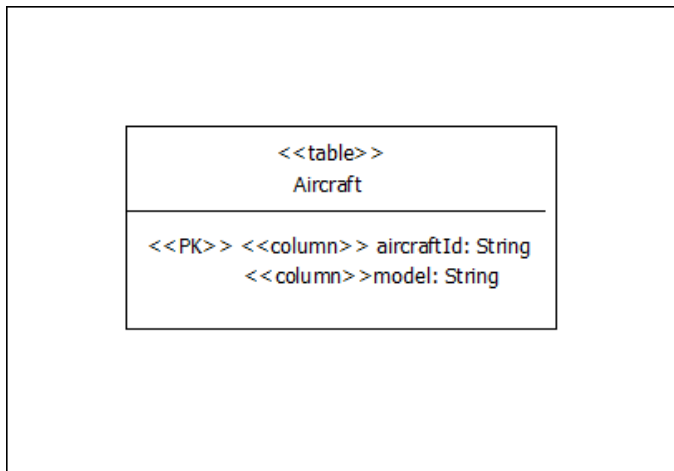


Figure: Aircraft table



# Representing inheritance

```
public class Aircraft {  
  @Id  
  String aircraftId;  
  String model;  
  . . .  
}
```

# Representing inheritance

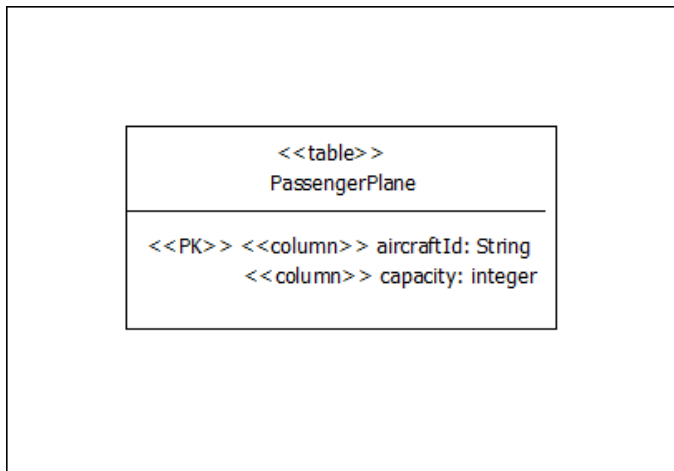


Figure: Passenger plane table

# Representing inheritance

```
public class PassengerPlane {  
    @Id  
    String aircraftId;  
    int capacity;  
    . . .  
}
```

# Representing inheritance

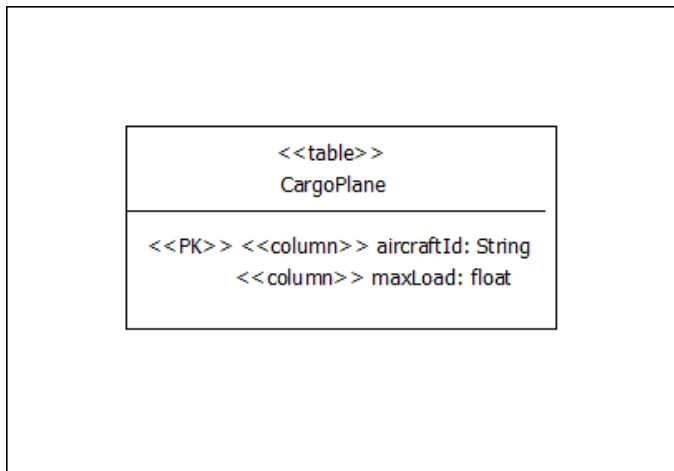


Figure: Cargo plane table

# Representing inheritance

```
public class CargoPlane {  
    @Id  
    String aircraftId;  
    float maxLoad;  
    . . .  
}
```

- Java persistence API queries
- Language Integrated Queries (LINQ)
- Comprehension queries
- Lambda queries
- Interfacing relational database
- Data context

```
@NamedQuery(  
  name= "findCheapStocks",  
  query ="SELECT s FROM Stock  
  WHERE s.price < 100"  
)
```

```
@PersistentContext  
EntityManager em;  
List cheapStocks = em.createNamedQuery(  
    findCheapStocks.getResultList());
```



```
System.Collections.IEnumerator  
System.Collections.Generic.IEnumerator<T>
```

```
class EnumeratorClass  
    // implements IEnumerator or IEnumerator<T>  
{  
    public IteratorVariableType Current { get {...} }  
    public bool moveNext() {...}  
}
```

System.Collections.IEnumerable  
System.Collections.Generic.IEnumerable<T>

```
class EnumerableClass  
    // implements IEnumerable or IEnumerable<T>  
{  
    public Enumerator GetEnumerator() {...}  
}
```

```
class Stock {  
    String stockId;  
    float price;  
    // other fields  
    // constructor  
    public String StockId  
    {  
        get { return stockId; }  
    }  
    public float Price  
    {  
        get { return price; }  
        set { price = value; }  
    }  
    // other properties  
}
```

```
IEnumerable<String> query =  
    from s in stocks  
    where s.Price < 100  
    orderby s.Price  
    select s.StockId;
```

```
foreach (String s in query) Console.WriteLine (s);
```

```
IEnumerable<String> query = stocks  
    .Where (s  $\Rightarrow$  s.Price < 100)  
    .OrderBy (s  $\Rightarrow$  s.Price)  
    .Select (s  $\Rightarrow$  s.StockId);
```

```
class StockPrice {  
  String stockId;  
  float price;  
  public String StockId;  
  {get}  
  public float Price;  
  { get and set }  
  // . . .  
}
```

```
IEnumerable<StockPrice> =  
  from s in stocks  
  select new StockPrice  
    { StockId = s.StockId;  
      Price= s.Price;  
    }  
  where s.Price < 100;
```

```
IEnumerable<String> =  
  from s in stocks  
  where s.Price()  $\geq$   
    Max(from f in stocks  
      where s.Company=f.Coompany  
      select f.Price)  
  select s.StockId;
```



```
create table Stock  
(  
    StockId varchar(4) not null primary key,  
    Price float  
)
```

```
[Table]  
public class Stock  
{  
    [Column(IsPrimaryKey=true)]  
    public String stockId;  
    [Column]  
    public float price;  
}
```

```
DataContext dataContext = new DataContext ("connection string");  
Table<Stock> stocks = dataContext.getTable <Stock>();  
IQueryable<String> query =  
    from s in stocks  
    where s.StockId="SP500"  
    orderby s.StockId  
    select s.StockId;  
  
foreach (String s in query) Console.WriteLine(s);
```