Daffodil International University

Assignment

Course title: Software Engineering Project II (Web Programming)
Course code: SWE 332

SUBMITED TO :

SK. Fazlee Rabby

Lecturer
Department of Software Engineering
Daffodil International University

SUBMITED BY :

MD. Bokhtiar Alam

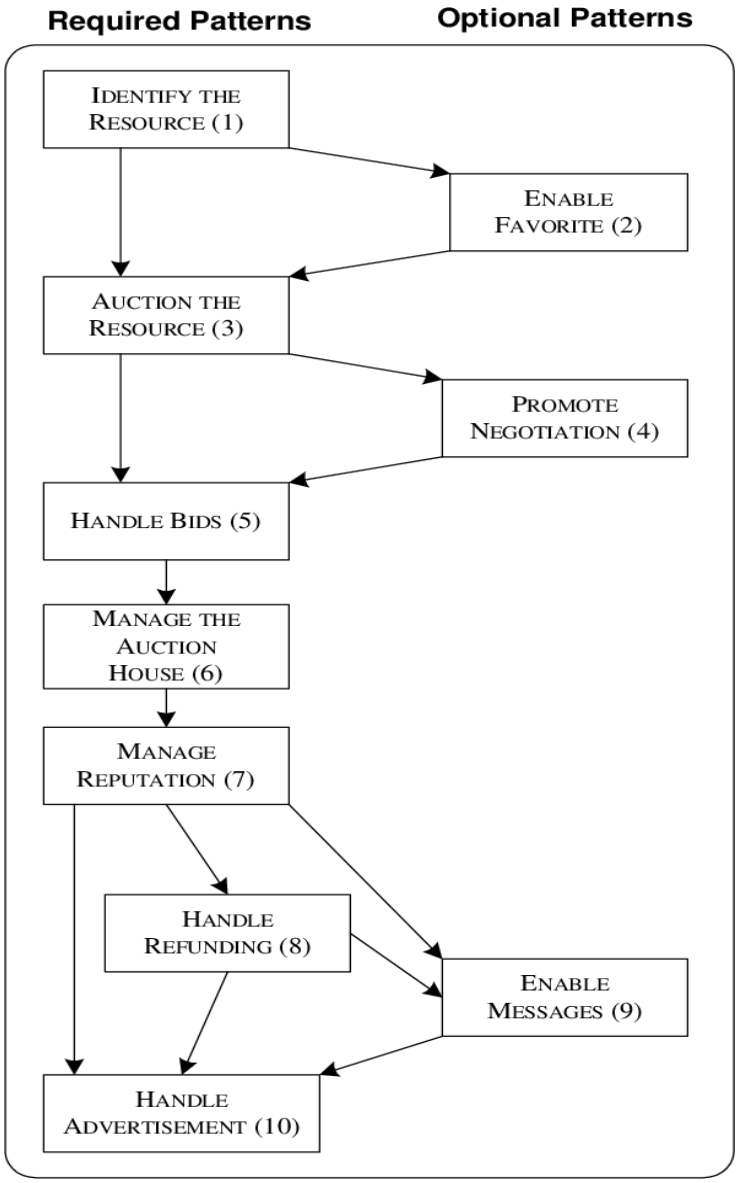
ID : 171-35-172

PC(A)

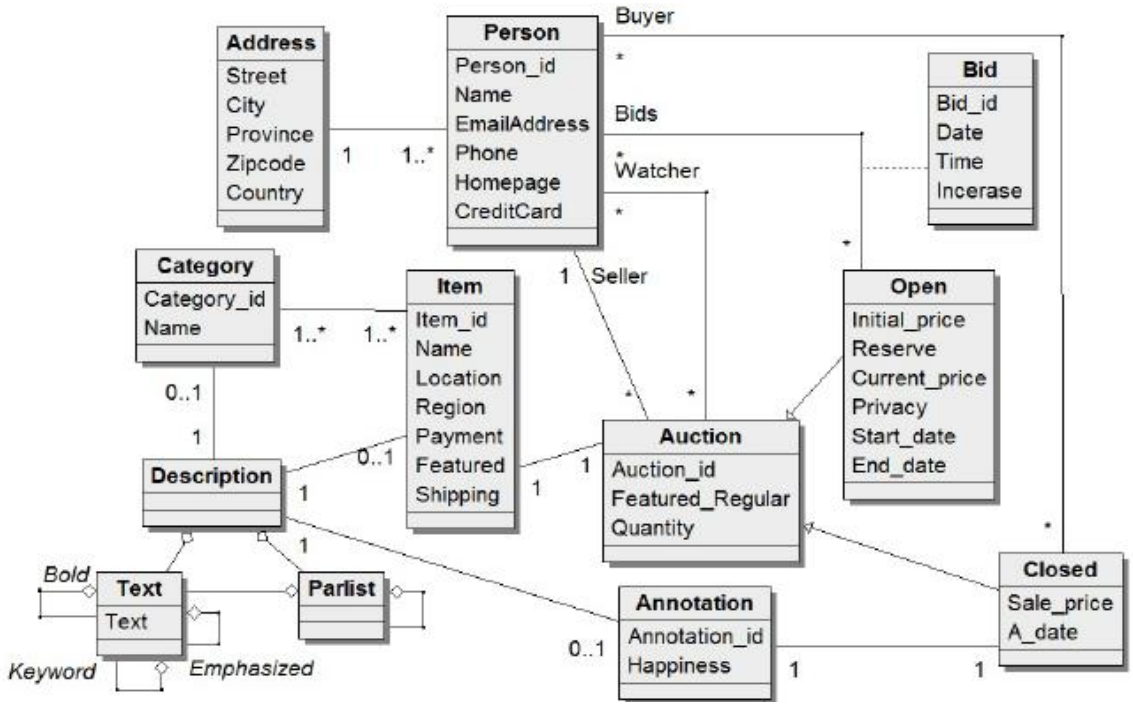
Department of Software Engineering

Daffodil International University

Finding the Right Design Pattern :



Draw Class Diagram :



Implement the code using Java :

```
package akka.persistence.multidc.javadsl;

 import java.io.Serializable;

import java.time.Instant;

import java.util.Collections;

import java.util.HashSet;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import com.typesafe.config.Config;

import com.typesafe.config.ConfigFactory;

import akka.actor.ActorRef;

import akka.actor.ActorSystem;

import akka.actor.Props;

import akka.japi.JAPI;

import akka.persistence.cassandra.testkit.CassandraLauncher;
```

```
import akka.persistence.multidc.PersistenceMultiDcSettings;
import akka.testkit.javadsl.TestKit;
//#cassandra-hook
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;
import org.scalatest.junit.JUnitSuite;
import akka.persistence.multidc.testkit.CassandraLifecycle;
//#disable-replication
import akka.persistence.multidc.testkit.PersistenceMultiDcTestKit;
//#disable-replication
//#cassandra-hook
import scala.concurrent.duration.Duration;
import static org.junit.Assert.assertEquals;
//#cassandra-hook
public class AuctionExampleTest extends JUnitSuite {
//#cassandra-hook
  //#auction-commands
  static class Bid implements Serializable {
    final String bidder;
    final int offer;
    final Instant timestamp;
    final String originDc;
    Bid(String bidder, int offer, Instant timestamp, String originDc) {
      this.bidder = bidder;
      this.offer = offer;
      this.timestamp = timestamp;
      this.originDc = originDc;
    }
    Bid withOffer(int offer) {
      return new Bid(bidder, offer, timestamp, originDc);
```

```
  }
}

// commands
interface AuctionCommand extends Serializable {}
static class OfferBid implements AuctionCommand {
  final String bidder;
  final int offer;
  public OfferBid(String bidder, int offer) {
    this.bidder = bidder;
    this.offer = offer;
  }
}
// An auction coordinator needs to schedule this event to each replica
static class Finish implements AuctionCommand {
  static final Finish INSTANCE = new Finish();
  private Finish() {}
}
static class GetHighestBid implements AuctionCommand {
  static final GetHighestBid INSTANCE = new GetHighestBid();
  private GetHighestBid() {}
}
static class IsClosed implements AuctionCommand {
  static final IsClosed INSTANCE = new IsClosed();
  private IsClosed() {}
}
// Internal, should not be sent from the outside
private static class Close implements AuctionCommand {
  static final Close INSTANCE = new Close();
  private Close() {}
}
```

```
//#auction-commands
//#auction-events
// events
interface AuctionEvent extends Serializable {}
static class BidRegistered implements AuctionEvent {
  final Bid bid;
  public BidRegistered(Bid bid) {
    this.bid = bid;
  }
}
static class AuctionFinished implements AuctionEvent {
  final String atDc;
  public AuctionFinished(String atDc) {
    this.atDc = atDc;
  }
}
static class WinnerDecided implements AuctionEvent {
  final String atDc;
  final Bid winningBid;
  final int highestCounterOffer;
  public WinnerDecided(String atDc, Bid winningBid, int highestCounterOffer) {
    this.atDc = atDc;
    this.winningBid = winningBid;
    this.highestCounterOffer = highestCounterOffer;
  }
}
//#auction-events

//#auction-state
static class AuctionState {
  final boolean stillRunning;
```

```
  final Bid highestBid;
  // in ebay style auctions, we need to keep track of current highest counter offer
  final int highestCounterOffer;
  final Set<String> finishedAtDc;
  AuctionState(boolean stillRunning, Bid highestBid, int highestCounterOffer, Set<String>
finishedAtDc) {
    this.stillRunning = stillRunning;
    this.highestBid = highestBid;
    this.highestCounterOffer = highestCounterOffer;
    this.finishedAtDc = finishedAtDc;
  }
  AuctionState withNewHighestBid(Bid bid) {
    assert(stillRunning);
    assert(isHigherBid(bid, highestBid));
    return new AuctionState(stillRunning, bid, highestBid.offer, finishedAtDc); // keep last highest bid
around
  }

  AuctionState withTooLowBid(Bid bid) {
    assert(stillRunning);
    assert(isHigherBid(highestBid, bid));
    return new AuctionState(stillRunning, highestBid, Math.max(highestCounterOffer, bid.offer),
finishedAtDc);
  }
  static Boolean isHigherBid(Bid first, Bid second) {
    return first.offer > second.offer ||
      (first.offer == second.offer && first.timestamp.isBefore(second.timestamp)) || // if equal, first
one wins
      // If timestamps are equal, choose by dc where the offer was submitted
      // In real auctions, this last comparison should be deterministic but unpredictable, so that
submitting to a
      // particular DC would not be an advantage.
```

```
      (first.offer == second.offer && first.timestamp.equals(second.timestamp) &&
first.originDc.compareTo(second.originDc) < 0);
  }
  AuctionState addFinishedAtDc(String dc) {
    Set<String> s = new HashSet<>(finishedAtDc);
    s.add(dc);
```