**SED 2022 Collaborative Solution**

**Q1**

**a.**

**TEST**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**@Test**   
 **public** **void** **bidsDuringAuctionChargeAccordinglyIfBidWinning**() {   
 context.checking(**new** Expectations( {{

exactly(**1**).of(paymentSystem).charge(**10**, "Alice");

}});   
  
 AuctionManager auctionManager = **new** AuctionManager("shoes", **0**, paymentSystem);

auctionManager.bid(**10**, "Alice");   
 }

**IMPLEMENTATION**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

AuctionManager.java

**package** ic.doc;   
import **java.util.ArrayList**;   
import **java.util.List**;   
  
public **class** **AuctionManager** {

**private** Pair<String, Integer> currentItem;   
 **private** PaymentSystem paymentSystem;

**public** **AuctionManager**(String currentItem, Integer currentBid, PaymentSystem paymentSystem) {   
  
 **this**.currentItem = **new** Pair<>(currentItem, currentBid);   
 **this**.paymentSystem = paymentSystem;   
 }  
  
 **public** **boolean** **bid**(Integer amount, String bidder) {   
  
 **if** (amount > currentItem.getSecond()) {   
 paymentSystem.charge(amount, bidder);   
  
 **return** **true**;   
 }   
  
 **return** **false**;   
 }   
}

PaymentSystem.java

**package** ic.doc;   
  
public **interface** **PaymentSystem** {   
 **void** **charge**(Integer amount, String person);   
}

Pair.java

**package** ic.doc;   
  
import **java.util.Objects**;   
  
public **class** **Pair**<T, T1> {   
  
 **private** T first;   
 **private** T1 second;   
   
 **public** **Pair**(T first, T1 second) {   
  
 **this**.first = first;   
 **this**.second = second;   
 }  
  
 **public** T **getFirst**() { **return** first; }   
  
 **public** **void** **setFirst**(T first) { **this**.first = first; }   
  
 **public** T1 **getSecond**() { **return** second; }   
  
 **public** **void** **setSecond**(T1 second) { **this**.second = second; }   
}

**b.**

**TEST**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**@Test**   
  
 **public** **void** **preAuctionInitializationBidsAreRejected**() {   
  
 context.checking(**new** Expectations() {{

exactly(**0**).of(paymentSystem).charge(**10**, "Alice");

}});   
  
 AuctionManager auctionManager = **new** AuctionManager("shoes", **0**, paymentSystem);   
  
 auctionManager.bid(**10**, "Alice");   
 }

**IMPLEMENTATION**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

AuctionManager.java

package ic.doc;

import java.util.ArrayList;

import java.util.List;

public class AuctionManager {

private Pair<String, Integer> currentItem;

private PaymentSystem paymentSystem;

private boolean auctionActive;

public AuctionManager(String currentItem, Integer currentBid, PaymentSystem paymentSystem) {

this.currentItem = new Pair<>(currentItem, currentBid);

this.paymentSystem = paymentSystem;

auctionActive = false;

}

public void activateAuction() { auctionActive = true; }

public void endAuction() { auctionActive = false; }

public boolean bid(Integer amount, String bidder) {

if (amount > currentItem.getSecond() && auctionActive) {

paymentSystem.charge(amount, bidder);

return true;

}

else return false;

}

}

**c.**

TEST

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

@Test

public void auctionItemDispatchedToWinner() {

Dispatcher dispatcher = new Dispatcher();

AuctionManager auctionManager = new AuctionManager("shoes", 0, paymentSystem, dispatcher);

dispatcher.setClient(auctionManager);

auctionManager.activateAuction();

auctionManager.bid(10, "Alice");

auctionManager.endAuction();

assertEquals(dispatcher.itemDispatched("shoes", "Alilce"), true);

}

IMPLEMENTATION

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

AuctionManager.java

package ic.doc;

import java.util.ArrayList;

import java.util.List;

public class AuctionManager {

private Pair<String, Integer> currentItem;

private String topBidder;

private PaymentSystem paymentSystem;

private boolean auctionActive;

private Dispatcher dispatcher;

public AuctionManager(String currentItem, Integer currentBid, PaymentSystem paymentSystem, Dispatcher dispatcher) {

this.currentItem = new Pair<>(currentItem, currentBid);

this.paymentSystem = paymentSystem;

auctionActive = false;

this.dispatcher = dispatcher;

}

public void activateAuction() { auctionActive = true; }

public void endAuction() {

auctionActive = false;

dispatcher.dispatch(currentItem.getFirst(), topBidder);

}

public boolean bid(Integer amount, String bidder) {

if (amount > currentItem.getSecond() && auctionActive) {

paymentSystem.charge(amount, bidder);

topBidder = bidder;

return true;

}

else return false;

}

}

Dispatcher.java

package ic.doc;

import java.util.ArrayList;

import java.util.List;

public class Dispatcher {

AuctionManager auctionManager;

private List<Pair<String, String>> itemsDispatched = new ArrayList<>();

public Dispatcher() {

}

public void setClient(AuctionManager auctionManager) {

this.auctionManager = auctionManager;

}

public void dispatch(String item, String person) {

itemsDispatched.add(new Pair(item, person));

}

public boolean itemDispatched(String item, String person) {

return itemsDispatched.contains(new Pair<>(item, person));

}

}

Pair.java

package ic.doc;

import java.util.Objects;

public class Pair<T, T1> {

private T first;

private T1 second;

public Pair(T first, T1 second) {

this.first = first;

this.second = second;

}

public T getFirst() { return first; }

public void setFirst(T first) { this.first = first; }

public T1 getSecond() { return second; }

public void setSecond(T1 second) { this.second = second; }

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Pair<?, ?> pair = (Pair<?, ?>) o;

return first.equals(pair.first) && second.equals(pair.second);

}

@Override

public int hashCode() {

return Objects.hash(first, second);

}

}

**d.**

TEST

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

@Test

public void sellerAndLosersReceiveOwedMoney() {

Dispatcher dispatcher = new Dispatcher();

AuctionPaymentSystem auctionPaymentSystem = new AuctionPaymentSystem();

AuctionManager auctionManager = new AuctionManager("shoes", 0, auctionPaymentSystem, dispatcher);

dispatcher.setClient(auctionManager);

auctionManager.activateAuction();

auctionManager.bid(10, "Alice");

auctionManager.bid(40, "Bertha");

auctionManager.bid(60, "Clive");

auctionManager.bid(30, "Randy");

auctionManager.endAuction();

assertEquals(auctionPaymentSystem.hasCharged(60, "Clive"), true);

assertEquals(auctionPaymentSystem.hasPaid(60, "SELLER"), true);

}

IMPLEMENTATION

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

AuctionManager.java

package ic.doc;

import java.util.HashMap;

import java.util.Map;

public class AuctionManager {

private Pair<String, Integer> currentItem;

private String topBidder;

private PaymentSystem paymentSystem;

private boolean auctionActive;

private Dispatcher dispatcher;

private Map<Integer, String> owed;

public AuctionManager(String currentItem, Integer currentBid, PaymentSystem paymentSystem, Dispatcher dispatcher) {

this.currentItem = new Pair<>(currentItem, currentBid);

this.paymentSystem = paymentSystem;

auctionActive = false;

this.dispatcher = dispatcher;

owed = new HashMap<>();

}

public void activateAuction() {

auctionActive = true;

}

public void endAuction() {

auctionActive = false;

dispatcher.dispatch(currentItem.getFirst(), topBidder);

owed.remove(topBidder);

owed.put(currentItem.getSecond(), topBidder);

}

public void payOwed() {

for (Map.Entry<Integer, String> entry : owed.entrySet()) {

paymentSystem.pay(entry.getKey(), entry.getValue());

}

}

public boolean bid(Integer amount, String bidder) {

if (amount > currentItem.getSecond() && auctionActive) {

paymentSystem.charge(amount, bidder);

topBidder = bidder;

owed.put(amount, bidder);

return true;

}

else return false;

}

}

PaymentSystem.java

package ic.doc;

public interface PaymentSystem {

public void charge(Integer amount, String person);

public void pay(Integer amount, String person);

}

AuctionPaymentSystem.java

package ic.doc;

import java.util.ArrayList;

import java.util.List;

public class AuctionPaymentSystem implements PaymentSystem {

private List<Pair<Integer, String>> bodiesCharged;

private List<Pair<Integer, String>> bodiesPaid;

public AuctionPaymentSystem() {

bodiesCharged = new ArrayList<>();

bodiesPaid = new ArrayList<>();

}

@Override

public void charge(Integer amount, String person) {

bodiesCharged.add(new Pair<>(amount, person));

}

@Override

public void pay(Integer amount, String person) {

bodiesPaid.add(new Pair<>(amount, person));

}

public boolean hasCharged(Integer amount, String person) {

return bodiesCharged.contains(new Pair<>(amount, person));

}

public boolean hasPaid(Integer amount, String person) {

return bodiesCharged.contains(new Pair<>(amount, person));

}

}

**Bit more compact way of representing Auction Manager and its relevant tests. Interfaces are pretty trivial:**

public class AuctionManager {  
  
 private final PaymentSystem paymentSystem;  
 private final Dispatcher dispatcher;  
 private Person seller;  
 private final Map<Person, Integer> biddingHistory = new HashMap<>();  
 private int highestBid = 0;  
 private Person highestBidder;  
 private AuctionItem item;  
  
 public AuctionManager(PaymentSystem paymentSystem, Dispatcher dispatcher) {  
  
 this.paymentSystem = paymentSystem;  
 this.dispatcher = dispatcher;  
 }  
  
 public void startAuction(AuctionItem item, Person seller) {  
 this.seller = seller;  
 this.item = item;  
 }  
  
 public BidResponse bid(int amount, Person bidder) {  
 if (highestBid < amount) {  
 paymentSystem.charge(amount, bidder);  
 if (highestBidder != null) {  
 biddingHistory.put(highestBidder, highestBid); // No test yet to test if duplicate bidders  
 }  
 highestBid = amount;  
 highestBidder = bidder;  
 return *BID\_ACCEPTED*;  
 } else {  
 return *BID\_TOO\_LOW*;  
 }  
 }  
  
 public void endAuction() {  
 // pay the seller  
 paymentSystem.pay(highestBid, seller);  
 // refund unsuccessful bids  
 for (Entry<Person, Integer> bid : biddingHistory.entrySet()) {  
 paymentSystem.pay(bid.getValue(), bid.getKey());  
 }  
 // dispatch item to winning bid  
 dispatcher.dispatch(item, highestBidder);  
 }  
}

Q2

**a.**

Singleton Pattern

private MediaLibrary()

...

private static MediaLibrary instance;

...

synchronized public static MediaLibrary getInstance() {

if (instance == null) {

instance = new MediaLibrary();

}

return instance;

}

**Part b & c:**

private final PlaybackEventLogInterface playbackEvents;  
private final MediaLibraryInterface mediaLibrary;  
private final TimeService timeService;  
  
public VideoStreamer(MediaLibraryInterface mediaLibrary, TimeService timeService,  
 PlaybackEventLogInterface playbackEvents) {  
 this.mediaLibrary = mediaLibrary;  
 this.timeService = timeService;  
 this.playbackEvents = playbackEvents;  
}

public VideoStreamer() {  
 this(MediaLibrary.getInstance(), LocalTime, new PlaybackEventLog());  
} // The original constructor with no arguments must be preserved as it is public

public List<Movie> getSuggestedMovies(User user) {  
 List<Movie> recommendations = mediaLibrary.recommendedMoviesFor(user);  
  
 ...  
}

public void stopStreaming(VideoStream stream) {  
 StreamTracker streamTracker = currentStreams.remove(stream);  
 LocalTime endTime = timeService.now();  
 long minutesWatched = ChronoUnit.*MINUTES*.between(streamTracker.startTime(), endTime);  
 if (minutesWatched > 15) {  
 playbackEvents.logWatched(streamTracker.user(), stream.movie());  
 }  
}

**TEST CLASS**

public class VideoStreamerTest {  
  
 @Rule  
 public JUnitRuleMockery context = new JUnitRuleMockery();  
 public MediaLibraryInterface mediaLibrary = context.mock(MediaLibraryInterface.class);  
 public TimeService timeService = context.mock(TimeService.class);  
 public PlaybackEventLogInterface playbackEventLog = context.mock(  
 PlaybackEventLogInterface.class);  
 public VideoStreamer streamer = new VideoStreamer(mediaLibrary, timeService, playbackEventLog);  
 public User user = new User("Adam", 9);  
 public final List<Movie> defaultMovies = List.*of*(  
 new Movie("Jurassic Park",  
 "",  
 420,  
 List.*of*(new Actor("David Attenborough")),  
 Set.*of*(Genre.*ADVENTURE*),  
 List.*of*(Oscar.*forBest*("Visual Effects")),  
 *PARENTAL\_GUIDANCE*  
));  
  
 @Test  
 public void allowsUserToStreamSuggestedMovies() {  
  
 context.checking(new Expectations() {{  
 exactly(1).of(mediaLibrary).recommendedMoviesFor(user);  
 will(*returnValue*(defaultMovies));  
 exactly(1).of(timeService).now();  
 will(*returnValue*(LocalTime.*now*()));  
 ignoring(playbackEventLog);  
 }});  
  
 List<Movie> movies = streamer.getSuggestedMovies(user);  
 VideoStream stream = streamer.startStreaming(movies.get(0), user);  
 streamer.stopStreaming(stream);  
 }  
  
 @Test  
 public void aFilmWatchedForMoreThanFifteenMinutesIsAddedToALog() {  
 Movie movie = defaultMovies.get(0);  
 context.checking(new Expectations() {{  
 exactly(1).of(timeService).now();  
 will(*returnValue*(LocalTime.*now*().plus(20, ChronoUnit.*MINUTES*)));  
 exactly(1).of(playbackEventLog).logWatched(user, movie);  
 }});  
  
 VideoStream stream = streamer.startStreaming(movie, user);  
 streamer.stopStreaming(stream);  
 }  
}