package player;

import java.util.Date;
import player.PlayerI.*;
import player.playeragent.*;
import edu.neu.ccs.demeterf.demgen.lib.List;
import gen.*;

/** Takes the Player’s Turn */
public class PlayerRunner{
    private Player player;
    private PlayerI fourthReich;

    public PlayerRunner(String pid, PlayerI play){
        fourthReich = play;
        player = new Player(new PlayerID(Integer.parseInt(pid)), play.getName());
    }

    /** Forwarded from Static to Dynamic */
    public void main(){
        List<Transaction> trans =
            buyOrReofferDerivative().
            .append(createDerivative()).
            .append(deliverRawMaterial()).
            .append(finishProduct())));
        Date stime = new Date();
        Util.commitTransaction(new PlayerTransaction(player,trans));
        Date etime = new Date();
        System.out.println("total time: " + (etime.getTime() - stime.getTime()));
    }

    /** This one shouldn’t need to be changed */
    public ReofferAgent getReofferAgent(){ return new ReofferAgent(); }

    /** Wrap a Derivative into a Transaction of the given TransactionType */
    private class TransWrap extends List<Map<Derivative, Transaction>>{
        private TransactionType type;
        public TransWrap(TransactionType t){ type = t; }
        public Transaction map(Derivative d){ return new Transaction(type, d); }
    }

    /** Creates a derivative */
    private List<Transaction> createDerivative(){
        Derivative d;
        List<Transaction> lot = List.<Transaction>create();
        for(int i = 0; i < 5; i++){
            d = fourthReich.getCreateAgent().createDerivative(player, Util.existingTypes());
            lot = lot.append(new Transaction(new Create(), d));
        }
        return lot;
        /*
        * For Next week
        * for(int i = 0; i < 5; i++){
        *    if(i < 3){
        *        d = fourthReich.getCreateAgent().createDerivative(player, Util.existingTypes());
        *        lot = lot.append(new Transaction(new Create(), d));
        *    } else{
        *        d = fourthReich.getCreateAgent().createDerivative(player, Util.existingTypes());
        *        lot = lot.append(new Transaction(new Create(), d));
        *    }
        * }
        * return lot;
        */
    }

    /** Buys a derivative from the sale stores or reoffers all of them */
    private List<Transaction> buyOrReofferDerivative(){
        List<Derivative> forSale = Util.forSale(player.id);
        if(forSale.isEmpty())return List.create();
```java
double account = Util.getAccount(player);
List<Derivative> bought = fourthReich.getBuyAgent().buyDerivatives(forSale, account);

if(!bought.isEmpty())
    return bought.map(new TransWrap(new Buy()));
return reofferAll(forSale, player.id);

/** Returns a list of reoffered derivatives */
private List<Transaction> reofferAll(List<Derivative> forSale, PlayerID pid) {
    return Util.uniquelyTyped(forSale).map(new PriceReducer(getReofferAgent(), pid));
}

/** Reduces an individual Derivative using the ReofferAgent */
private class PriceReducer extends List.Map<Derivative, Transaction>{
    ReofferAgent agent;
    PlayerID pid;
    PriceReducer(ReofferAgent a, PlayerID p){ pid = p; agent = a; }

    public Transaction map(Derivative d){
        return new Transaction(new Reoffer(), agent.reofferDerivative(d, pid));
    }
}

/** Delivers raw material for the derivatives that need raw material */
private List<Transaction> deliverRawMaterial(){
    List<Derivative> needRM = Util.needRawMaterial(player);
    return needRM.map(new Deliverer(fourthReich.getDeliverAgent()));
}

/** Handles the calling of DeliverAgent */
private class Deliverer extends List.Map<Derivative, Transaction>{
    DeliverAgentI agent;
    Deliverer(DeliverAgentI a){ agent = a; }

    /** Call the DeliverAgent, and Wrap the Transaction */
    public Transaction map(Derivative d){
        return new Transaction(new Deliver(), agent.deliverRawMaterial(d));
    }
}

/** Finishes the derivatives that need finishing */
private List<Transaction> finishProduct(){
    List<Derivative> toFinish = Util.toBeFinished(player);
    return toFinish.map(new Finisher(fourthReich.getFinishAgent()));
}

/** Handles the calling of FinishAgent */
private class Finisher extends List.Map<Derivative, Transaction>{
    FinishAgentI agent;
    Finisher(FinishAgentI a){ agent = a; }

    /** Call the FinishAgent, and Wrap the Transaction */
    public Transaction map(Derivative d){
        return new Transaction(new Finish(), d.finish(agent.finishDerivative(d)));
    }
}
```