```
/* *********************************
 *   PlayerRunner.java
 *     Generic Player Main Methods
 * *********************************/
package player;

import player.PlayerI.*;
import player.playeragent.*;
import edu.neu.ccs.demeterf.demfgen.lib.List;
import gen.*;

/** Takes the Player's Turn */
public class PlayerRunner{
        private Player player;
        private PlayerI genericPlayer;

        public PlayerRunner(String pid, PlayerI play){
        genericPlayer = 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())));

        Util.commitTransaction(new PlayerTransaction(player,trans));
    }

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


    /** Wrap a Derivitive 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 der = genericPlayer.getCreateAgent().createDerivative(player, Util.existingTypes());

        return List.create(new Transaction(new Create(), der));
    }

    /** Buys a derivative from the sale stores or reoffers all of them */
    private List<Transaction> buyOrReofferDerivative(){
        List<Derivative> forSale = Util.forSale();
        if(forSale.isEmpty())return List.create();

        double account = Util.getAccount(player);
        List<Derivative> bought = genericPlayer.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(genericPlayer.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(genericPlayer.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)));
        }
    }
}
```