

```

/**
 * @author Harley Phung
 * Create a MarketBuyOrder for market maker to start transaction
 */
public class MarketBuyOrder extends Order implements MarketOrder, BuyOrder{
    /** the Stock symbol of the market maker */
    private char stockSymbol = ' ';

    /** The number of shares of the order */
    private int numShares = 0;

    /** The price per share the market maker sell */
    private double numPrice = 0.0;

    /**The MarketMaker instance */
    private Trader buyMarket = null;

    /**
     * A constructor that have stock symbol, number of share, price per share and
     trader instance as input
     * @param stockSymbol the symbol of the buy market order
     * @param numShares the number of shares of the buy market order
     * @param numPrice the price per share of the buy market order
     * @param buyMarket the instace of the trader in this buy market order
     */
    public MarketBuyOrder (char stockSymbol, int numShares, double numPrice, Trader
    buyMarket) {
        this.stockSymbol = stockSymbol;
        this.numShares = numShares;
        this.numPrice = numPrice;
        this.buyMarket = buyMarket;
    }

    /**
     * Returns the stock symbol of the market maker
     * @return stockSymbol the symbol of this buy market order
     */
    @Override
    public char getStockSymbol() {
        return this.stockSymbol;
    }

    /**
     * Returns the number of shares of the order
     * @return numShares the number of shares in this order
     */
    @Override
    public int getNumberShares() {
        return this.numShares;
    }

    /**
     * Changes the number of shares of the order
     * @param numShares new number of shares of the order
     */
    @Override
    public void setNumberShares(int numShares) {
        this.numShares = numShares;
    }
}

```

```

/**
 * Returns desired price per share of the order
 * @return numPrice the price per share of the order
 */
@Override
public double getPrice() {
    return this.numPrice;
}

/**
 * An override method that override the isAllOrNone method
 * @return false marketMaker never trade all
 */
@Override
public boolean isAllOrNone() {
    return false;
}

/**
 * An override method that override the isDayOrder method
 * @return true marketMaker always process in day order
 */
@Override
public boolean isDayOrder() {
    return true;
}

/**
 * An override method that return instance of the market maker
 * @return buyMarket the instance of the market maker in this buy market order
 */
@Override
public Trader getTrader() {
    return this.buyMarket;
}

/**
 * toString method format the returned String
 */
@Override
public String toString() {
    return this.getStockSymbol() + ", " + this.getNumberShares() + ", " +
this.getPrice()
        + ", " + this.getTrader();
}

/**
 * An abstract equals method that compared the two trader's information.
 * @param p compare the trader
 * @return true if there's identical market buy order
 * @return false if there's no identical market buy order
 */
@Override
public boolean equals(Object p){
    if(p instanceof MarketBuyOrder) {
        MarketBuyOrder newBuyOrder = (MarketBuyOrder)p;
        if(this.getStockSymbol() == newBuyOrder.getStockSymbol())

```

```
        && this.getNumberShares() == newBuyOrder.getNumberShares()
        && this.getPrice() == newBuyOrder.getPrice()
        && this.getTrader() == newBuyOrder.getTrader()){
            return true;
        }
    }
    return false;
}
}
```