```
/**
 * @author Harley Phung
 * Create a MarketSellOrder for market maker to trade with trader
 */
public class MarketSellOrder extends Order implements MarketOrder, SellOrder{
    /** 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 sellMarket = null;

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

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

    /**
     * Returns the number of shares of the order
     * @return numShares  the number of shares of the sell market 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 sell market 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 sellMarket  the instance of the trader in this sell market order
     */
    @Override
    public Trader getTrader() {
        return this.sellMarket;
    }

    /**
     * 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 sell order
     * @return false  if there's no identical market sell order
     */
    @Override
    public boolean equals(Object p){
        if(p instanceof MarketSellOrder) {
            MarketSellOrder newSellOrder = (MarketSellOrder)p;
            if(this.getStockSymbol() == newSellOrder.getStockSymbol()
              && this.getNumberShares() == newSellOrder.getNumberShares()
```

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