Place candidate's barcoded sticker here.

This sheet must be stapled to the font of each candidates' submission.

I.T. PRACTICAL EXAMINATION 2012

Version:	**Final Marksheet (19 October 2012)**	**Prog. Lang.**	
Candidate:			
Marker:		**120**	**0**
Checksum:	0000-0000-0120	**Total**	**Pupil**

1.1	SELECT *✓ FROM tblWaiters✓ ORDER BY waiterName✓;	**3**	
1.2	SELECT tableID, tableGuests FROM tblTables✓ WHERE tblGuests = 1 OR✓ tblGuests >= 10✓; - 2 marks for correct condition, -1 for errors to a max of -2.	**3**	
1.3	SELECT menuDescription FROM tblMenuItems✓ WHERE menuDescription LIKE✓ '*chips*'✓; - can give marks for INSTR, penalise 1 mark for only 1 wildcard.	**3**	
1.4	SELECT menuDescription, menuSalesPrice – menuCostPrice✓ AS profit✓ FROM tblMenuItems WHERE menuCategory = "Drinks"✓;	**3**	
1.5	INSERT INTO tblWaiters✓ (waiterName, waiterPhone)✓ VALUES ("Busi", "083 469 9000")✓; - if inserted a key value, lose the mark assigned for fields	**3**	
1.6	SELECT menuDescription, (menuSalesPrice / menuCostPrice – 1)✓ * 100✓ AS MarkUp FROM tblMenuItems✓ ORDER BY (menuSalesPrice / menuCostPrice - 1)✓ * 100 DESC✓	**5**	
1.7	SELECT menuDescription, SUM✓(orderQuantity)✓ AS Quantity FROM tblMenuItems INNER JOIN tblOrders✓ ON tblOrders.orderMenuItemID = tblMenuItems.menuID✓ GROUP BY menuDescription✓; left join -1; WHERE joins are acceptable	**5**	
1.8	SELECT waiterName, COUNT✓(tableID)✓ AS tablesServed, AVG✓(tableAmountPaid)✓ AS avgAmountPaid FROM tblWaiters INNER JOIN tblTables✓ ON tblWaiters.waiterID = tblTables.tableWaiterID✓ GROUP BY waiterName✓; COUNT(tableID) can be COUNT(*); LEFT JOIN -1; WHERE joins acceptable	**7**	
1.9	SELECT waiterName, SUM(orderQuantity)*10✓ AS Prize FROM tblWaiters✓ INNER JOIN (tblTables✓ INNER JOIN (tblMenuItems✓ INNER JOIN tblOrders ON tblMenuItems.menuID = tblOrders.orderMenuItemID) ON tblTables.tableID = tblOrders.orderTableID) ON tblWaiters.waiterID = tblTables.tableWaiterID✓✓ WHERE menuDescription LIKE "*Giant Burger*"✓ GROUP BY waiterName✓; Using ID 2 inner joins (e.g. using 'id=x or id=y') is fine if answer meets what is asked. Check against supplied output.	**8**	
2	Class header is correct✓; PROPERTIES: all private✓, all appropriate data types✓, all named correctly✓; CONSTRUCTOR: method header is correct✓, assignments are correct✓; GETTERS: all getters correct✓✓ (-1 per error to a max of 2); METHODS: setter is correct✓; changeQuantity has correct header✓, increase✓; toString has correct header✓, formatting & fields✓; If utilise different protected or public instead of private to make later questions easier, a mark will be lost at beginning only; don't be too strict on formatting (spaces / tabs etc.); field names must bear relation to what was asked, not "x" or "y".	**13**	
3.1	Class header is correct with extend✓	**1**	
3.2	**Properties:** both private✓, both double✓, both named appropriately✓ As long as data type can handle decimals i.e. real NO MARKS FOR PROTECTED, but "carry the error" and don't re-penalise later for no method calls.	**3**	
3.3	**Constructor:** header is correct✓✓ (-1 per error to a max of 2), calls parent constructor✓, assignments are correct✓. If super not called -1 mark; if re-declares parent properties (above) then penalise here: either super-constructor call is irrelevant or not present.	**4**	

3.4	**mustOrder method:** method header is correct✓, if statement with correct condition (getQuantity () < minimumLevel)✓✓ (-1 per error to a max of 2), return true else, return false✓. If code added to decrement methods functionality mark will be deducted. If has no effect on execution of method no marks deducted; return a boolean condition instead of using "if" is also acceptable: "return getQ < minLev";	**4**	
3.5	**getOrderAmount method:** method header is correct✓, correct calculation for return maximumLevel – getQuantity ()✓✓ (-1 per error to a max of 2); if used "int" as property data-type, don't repenalise here.	**3**	
4.1	Class header is correct✓	**1**	
4.2	**Properties:** both are declared private✓, correct data type for each (StockItem array✓, int✓), both initialised (array of 100, count ~~= 0~~) ✓ No marks for protected or public; penalise here and don't carry through. "count" mark for declaration, not initialisation.	**4**	
4.3	**Constructor:** method header correct✓, open file for reading✓, indefinite loop✓, correct looping condition✓, parse line on "#"✓, "if" determines object correctly✓, correctly create a StockItem object with parameters✓, correctly create a StockItemOrder with parameters✓, increment counter✓, read in a new line in the loop✓ In Delphi "open file" means "everything that needs to be done to read from a file"; note that Java memo has class name typo.	**10**	
4.4	**getStockList:** method header correct✓, initialise a temporary variable✓, appropriate ~~for~~ loop✓, concatenate✓ the toString✓ with a newline✓, return concatenated variable (no mark for "")✓ - If protected or public used earlier mark deducted earlier do not penelise here if produces desired output	**7**	
4.5	**getOrderingList:** method header correct✓, for loop to loop through each element✓, if-statement to check object type✓, type-casting✓, if-statement to check for order✓, concatenate correct fields (getDescription () + ": " + getOrderAmount() + " " + getUnit ()) to return var with a newline✓✓ (-1 for errors to a max of 2), return✓ - formatting must be similar to requirements, doesn't have to be precise.	**8**	
4.6	**findStockItem:** method header correct✓, for-loop to iterate through all records✓, compare to search string✓, return found object✓, return null if none✓. If method type is int or String, null cannot be returned: lose 2 marks for header and null return; there are no marks for case-sensitive testing, so don't penalise if not done.	**5**	
5.1	Class header correct✓	**1**	
5.2	Instantiate a StockManager object✓	**1**	
5.3	Print both headings✓, print stock list✓, print ordering list✓ - if output was only printed after update, award these marks for that code; they lose the "repeat output" mark	**3**	
6.1	**updateStockLevels:** open file for reading and use indefinite loop with correct condition✓, parse text✓✓, find stock item✓, some check for "used"✓ and reduce✓, some check for "bought" ✓ and increase✓, otherwise set level✓ (check conditions can be in any order and use any working method), read next line✓.	**10**	
6.2	Perform stock take✓, display info as before✓ - if, in GUI, buttons were provided instead of sequenced code, award this mark for "interface"; if output is done only once, do not award "display info as before mark".	**2**	