

APOLLO

B3 Object-oriented programming SL

Computer Science
First assessment 2027
2 questions | mark schemes

3. (a) **Award [4 max]**

creates / assigns memory to a new instance of the `SalesPerson` class;
identified by the variable `employee1`;
initialises `id` with value "A6457098";
declares (empty) sales array;

(b) **Award [2 max]**

`bonus1` belongs to the class / it will remain the same for all instances of the class / has the same memory for all instances;
`id` belongs to the object/instance / it will be different for each instance / be assigned different memory for each instance;

`id` is declared inside the constructor method;
`bonus1` is declared inside the class definition but outside any of the instance methods and constructor;

`id` gets created when an instance of the class is created;
`bonus1` is created when the program begins to execute;

changes made to instance variable `id` through one object will not reflect in another object;
changes made in the class variable `bonus1` will reflect in all objects;

(c) **Award [2 max]**

provides greater security for the values stored in these variables/for the code;
restricts access to such variables from outside of the class /only allows access through public / getter, setter methods;
(helps to) prevents accidental/unintended changes to the variables (from outside of the class);

(d) **Award [2 max]**

Award [1] for parameters;
Award [1] for adding the sales figure to the array;

```
public void addSales(double amount,int month)
{
    sales[month] = amount;
}
```

– 11 –

(e) Award [5 max]

Award [1] for a correct loop;
Award [1] for initializing highest;
Award [1] for the comparison;
Award [1] for updating highest;
Award [1] for returning highest;

```
public double calculateHighest()
{
    double highest = 0;
    for (int x = 0; x < 12; x++)
    {
        if (this.sales[x] > highest)
            highest = this.sales[x];
    }
    return highest;
}
```

(f) Award [5 max]

Award [1] for a loop and initialising and updating bonus variable;
Award [1] for a correct comparison using `calculateTotal` method;
Award [1] for a correct comparison using `calculateHighest` method;
Award [1] for a correct identification of `bonus1` and `bonus2`;
Award [1] for prints out the id (using getter) and bonus (allow values only);

```
public void bonus(SalesPerson[] employee)
{
    for (int x = 0; x < 10; x++)
    {
        int bonus = 0;
        if (employee[x].calculateTotal() > 300000)
            bonus = bonus + SalesPerson.bonus1;
        if (employee[x].calculateHighest() > 20000)
            bonus = bonus + SalesPerson.bonus2;
        System.out.println ("ID is " + employee[x].getId());
        System.out.println ("Bonus is " + bonus);
    }
}
```

Option B — Python

6. (a) Award [4 max]

creates / assigns memory to a new instance of the `SalesPerson` class;
 identified by the variable `employee1`;
 initialises `id` with value "A6457098";
 declares (empty) sales list;

(b) Award [2 max]

`bonus1` belongs to the class / it will remain the same for all instances of the class / has the same memory for all instances;
`self_id` belongs to the object/instance / it will be different for each instance / be assigned different memory for each instance;

`self._id` is declared inside the constructor/ the `__init__()` method;
`bonus1` is declared inside the class definition but outside any of the instance methods and constructor;

`self.id` variable gets created when an instance of the class is;
`bonus1` is created when the program begins to execute;

changes made to instance variable `self._id` through one object will not reflect in another object; changes made in the class variable `bonus1` will reflect in all objects;

(c) Award [2 max]

provides greater security for the values stored in these variables/for the code;
 restricts access to such variables from outside of the class /only allows access through public / getter, setter methods;
 (helps to) prevents accidental/unintended changes to the variables (from outside of the class);

(d) Award [2 max]

Award [1] for parameters;

Award [1] for adding the sales figure to the list;

```
def add_sales(self, amount):
    self._sales.append(amount)
```

```
def add_sales(self, amount):
    self._sales = self._sales + [amount]
```

– 17 –

(e) **Award [5 max]**

Award [1] for a correct loop;
Award [1] for initializing highest;
Award [1] for the comparison;
Award [1] for updating highest;
Award [1] for returning highest;

```
def calculate_highest(self):
    highest = 0
    for x in range (12):                # ... range (0,12)
        if self._sales[x] > highest:
            highest = self._sales[x]
    return highest
```

```
def calculate_highest(self):            1
    highest = 0
    for x in range (len(self._sales)):
        if self._sales[x] > highest:
            highest = self._sales[x]
    return highest
```

```
def calculate_highest(self):
    highest = 0
    for sales_figure in self._sales:
        if sales_figure > highest:
            highest = sales_figure
    return highest
```

(f) **Award [5 max]**

Award [1] for a loop and initialising and updating bonus variable;
Award [1] for a correct comparison using `calculate_total` method;
Award [1] for a correct comparison using `calculate_highest` method;
Award [1] for a correct identification of `bonus1` and `bonus2`;
Award [1] for prints out the id (using `getter`) and bonus (allow values only);

```
def calculate_bonus():
    for x in range(10):                # ... range (0,10)    ... range
        (len(self._sales))

        bonus = 0
        if employee[x].calculate_total() > 300000:
            bonus = bonus + SalesPerson.bonus1
        if employee[x].calculate_highest() > 20000:
            bonus = bonus + SalesPerson.bonus2
        print (f"Employee {employee[x].get_id()} has bonus ", bonus).
```