Option D — Object-oriented programming

10. (a) **`public`**;
Allows access to variables from outside of the class/unlimited access;

`protected`;
Allows access to variable from within the package (project) in which they are created/subclasses;

`final`;
Prevents variables from being modified;

`static`;
Refers to variables that act on the class as a whole (and not on individual objects);

Note:
- *Accept at most one example pertaining to methods.*
- *Do not accept two examples pertaining to the same modifier.* **[4]**

(b) The OOP feature shown in the constructors (accept the 2 signatures) is overloading (accept polymorphism);
The constructor methods have a different number / type of parameters / different parameters;
The method calling the constructor / compiler will determine which of these methods is selected;
By matching up with the parameters; **[4]**

(c) Silver; **[1]**

(d) Through the use of the (appropriate) array index / appropriate code description;

<u>Example:</u>
`Individual object = allVisits[individual object’s location]` **[1]**

(e)
```
public boolean isGold()
{
  return (statusNow.equals("Gold"); // allows =
}
```

Award marks as follows:
Signature;
Correct comparison (allow use of `getStatusNow()`);
Return (that matches the signature – allow FT);

Note:
- *Allow the equivalent use of IF/THEN statements.*
- *Do not accept parameters to be passed.*
- *Do not allow the use of `totalPoints`.* **[3]**

11. (a) *Award **[1]** for three compartments, **[1]** for correct + and –, and **[1]** for correct contents.*

Visits
- hotelCode: String - days: **int**
+ Visits(String, **int**) + getDays():**int**

***Note**:*
- *allow variations in the format, but must use + / –*
- *accept additional getters/setters, but the given content must be present.* **[3]**

(b) (i) m102; **[1]**

(ii) 0; **[1]**

(iii) 6; **[1]**

(c) *Example 1:*
```
public int calculateTotalPoints()
{
  int totalPoints = 0;
  for (int x = 0; x < y; x++)
  {
    totalPoints = totalPoints + allVisits[x].getDays();
  }
  totalPoints = totalPoints * 1000 + bonusPoints;
  return totalPoints;
}
```

*Award marks for **correctly** including the following:*
Signature + matching return;
Loop through the number of visits (y); // *do not allow length statements;*
Any use of `allVisits` array;
Correct update of `totalPoints` (with or without bonusPoints);
Inclusion of bonus points outside of the loop (or if the loop is absent);

Example 2:
```
public int calculateTotalPoints()
{
  int totalDays = 0;
    for (int x = 0; x < y; x++)
    {
      totalDays = totalDays + allVisits[x].getDays();
    }
    totalPoints = totalDays * 1000 + bonusPoints;
    return totalPoints;
}
```
[5]

(d)
```
public int daysMissing()
{
  int pointsNeeded = 0;
  int points;

  // convert present status to minimum number of days
  if (statusNow.equals("Silver"))      // allow = or use of
                                       // isSilver()
    pointsNeeded = 10000;
  else if (statusNow.equals("Gold"))   // allow = or use of
                                       // isGold()
    pointsNeeded = 50000;
  points = pointsNeeded - calculateTotalPoints();
  if (points > 0)  // might be negative
  {
      return points/1000;
  }
  else
  {
      return 0;
  }
}
```

Award marks for correctly including the following:
Signature + return of an integer;
All required declaration (initialization if needed);
Conversion to points/days for all 3 statuses // *bronze can be the default;*
Calculation of points missing (allow `getTotalPoints()` or the variable `totalPoints` for `calculateTotalPoints()`) ;
Convert to days (divide by 1000) // *allow if the conversion has already taken place;*
Returning calculated number; // *allow even if negative or if incorrect or without an if else clause;*
Returning 0 // *when the points needed would have been negative / when required points have already been gained;*

(e) A generic class / Status / Point / Bronze class can be used as a superclass;
Sub-classes can then be created (2 or 3) for the individual statuses;
Containing elements specific to them / overriding superclass methods;
And inheriting methods/variables required by all status levels from the superclass;

*Do not award more than **[2]** for a generic response.* **[4]**

(f) (In the Points class) `statusNow` = `statusNextYear`;
`bonusPoints` set to 0 (accept reset `bonusPoints`);
`totalPoints` set to 0 (accept reset `totalPoints`);
variable y (that counts the visits) needs to be set to 0;
`statusNextYear = Bronze` (accept reset `statusNextYear`);

Note: *Do not accept “array `allVisits` reinitialised to empty”.* **[3 max]**

12. *Award up to **[7 max]**. Note there are 9 marking points.*

An array of objects / 2 parallel arrays would be created / any other appropriate structure; // *Do not allow 2D array;*
Containing hotel codes and number of days;
Repeat/loop for each object in the `allVisits` array;
(i) Inspect the hotel ids and days stayed / find a matching hotel;
(ii) Update the array(s); // *see first note below*
(iii) By increasing the number of days (for the specific hotel);
Sort / search / look for / find; // *see first note below;*
The object / hotel code with the largest number of days;
Find the name of the hotel using the hotel code / Hotel class; **[7 max]**

Note:
- *mps can be awarded even if the wrong values are being updated or searched for (eg stays instead of days).*
- *For mp 7 only allow the use of "look for" and "find" if accompanied by a suitable description.*