Option D — Object-oriented programming

10. (a) The two constructors have different parameter sets (or equivalent);
Compiler can differentiate between the two;
The compiler will execute the constructor whose parameter set matches the arguments in the constructor call ;
This is an example of polymorphism/overloading; **[3]**

(b) Access is restricted to;
(methods that are part of) the same package / project; **[2]**

11. (a) *Award **[2]** for all 3 correct and **[1]** for any 2 correct.*
`routeCode` is `String`
`delay` is `int` *(**Note:** Allow any numeric)*
`weatherRelated` is `Boolean` **[2]**

(b) "`public Journey`";
Correct parameters *(**Note:** Allow FT from part (a))*;
Correct assignment statements;
***Note:** Allow any order. Allow absence of "`this`".*

```
public Journey(String a, int b, Boolean c)
{
  this.routeCode = a;
  this.delay = b;
  this.weatherRelated = c;
}
```
[3]

(c) T290;
10;
1;
***Note:** The punctuation (";") is not output.* **[3]**

12. (a) 4.5; **[1]**

(b) *Award marks as follows:*
Initializing variables used (e.g.total as a double and count as an integer);
Correct loop;
Correct comparison* (allow getJourney[x]);
Updating total*;
Updating count (in correct position);
Returning average;
Note: **If "get" methods are **not** used but otherwise correct, award **[1]** for these two points.*

```
public double averageDelay()
{
  int delayTotal = 0; // Allow total as a double with no
                      // casting below
  int count = 0;
  for (int x=0; x<numberOfJourneys; x++)
  {
    if (!journeyHistory[x].getWeatherRelated())
    {
         delayTotal = delayTotal +
                           journeyHistory[x].getDelay();
         count++;
    }
  }
  return double(delayTotal)/count;
}
```
[6]

13. (a) *Award **[1]** for each section (award **[2]** if correct except for +/-).*
Allow for slight variations of syntax.
Allow if constructer missing.
Allow public class Codes.

```
              Codes
- routeName : String
- routeCode : String
+Codes(a: String, b: String)
+getRouteName() : String
+getRouteCode() : String
```
[3]

(b) *Award marks as follows:*
Initialization of `maxDelay` to be a rogue value or 0, or 1st value;
Award {2 marks] for correct comparison (award 1 mark if weather related ignored);
Updating of `maxDelay`;
Updating of `maxCode`;
Both loops correct;
Searching for route code;
Return route name;

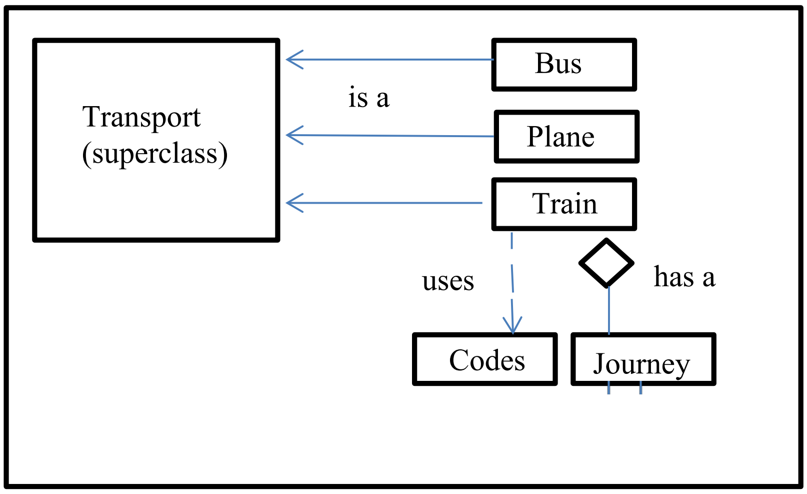
```
public String longestDelay(Codes [] c)
{
  String route = " ";
  String maxCode =" ";
  int maxDelay =-1;
  for (int x = 0; x < numberOfJourneys; x++)
  {
    if ((journeyHistory[x].getDelay() > maxDelay) &&
                  (!journeyHistory[x].getWeatherRelated()))
    {
      maxDelay = journeyHistory[x].getDelay();
      maxCode = journeyHistory[x].getRouteCode();
    }
  }
  for (int y=0; y<c.length; y++)
  {
    if (c[y] !=  null)
    {
      if (c[y].getRouteCode() == maxCode)
      {
        route = c[y].getRouteName();
      }
    }
  }
  return route;
}
```

[7]

14. (a) A super-class could be created (*eg* `Transport`);
With sub-classes Bus, Train and Plane;
Containing variables/methods common to all the different transport classes/examples;
The individual sub-classes can inherit these common attributes;
Whilst having properties particular to themselves/ can override; **[4]**

(b) (i) *Award **[1]** for a diagram showing five (or more) classes linked in some way.*
*Award **[1]** for each of the three different dependencies shown, either with the correctly shaped arrow or with a written description, up to **[3 max]**.*

The dependencies are:
Each of the train company, bus and airplane classes "is a" Transport class;
The train company class "has a" Journey;
The train company class "uses" the Codes class;



[4]

(ii) Changes in one class in a dependency will/may affect the other class(es) in this dependency;
This may cause programs using the second class not to function (correctly)/require modifications to the second class to avoid problems;
For example, if the variables in the `Codes` class changed type then the `TrainCompany` *etc* methods would have to be changed;

(Less dependencies lead to) reduced maintenance overheads;
As a programmer editing one class would not have to be concerned with other classes / allows programmers to focus just on the class they are writing; **[3]**

(c)
```
public String toString(Codes [] c)
{
  String d = companyName;
  double e = averageDelay();
  String f = longestDelay(c);
  String result = d + " : Average Delay = " + e + " :
                                   Longest Delay = " + f;
  return result;
}
```

Note: *Can be written in one line. Students may introduce validation on* `e`*.*
Award marks as follows:
Extracting company name;
Correct use of `averageDelay()` method;
Correct use of `longestDelay()` method;
Correct result line <u>either returned or output;</u>
*(**Note:** Ignore minor punctuation errors/missing parameter)*; **[4]**