Option D — Object-oriented programming

14. (a) *Award up to **[2 marks max]**.*
It is the method that instantiates/creates a new object;
It may be used to initialize variables (of the new object); ***[2 marks]***

(b) *Award **[1 mark]** for identifying an advantage.*
*Award **[1 mark]** for identifying an example.*
*Award **[1 mark]** for an elaboration.*

Example:
Polymorphism allows an external program to use the same method action on all subclasses;
By allowing overridden functions in child classes to add only the code that is needed for the unique processing of that sub-class;
In this example, the `getWeight()` method returns the weight of each piece of `RollingStock`. In the case of a wagon, the additional computation needed to add the weight of the cargo is added; ***[3 marks]***

(c) *Award **[1 mark]** for correct UML box.*
*Award **[1 mark]** for correct member variables.*
*Award **[1 mark]** for correct member functions.*

Train
-mEngines : Engine[] -mWagons : Wagon[] -mEngineCount : int -mWagonCount : int -mTrainNumber : int -mWeight : double
+Train(number : int) +addEngine(newEngine : Engine) +removeEngine() : Engine +addWagon(newWagon : Wagon) +removeWagon() : Wagon +getWeight() : double

[3 marks]

(d) *Award **[1 mark]** for correct function declaration line, including `public`.*
*Award **[1 mark]** for returning the correct value.*

Example:
```
public int getNumberOfWagons()
{
   return mWagonCount;
}
```
[2 marks]

(e) *Award **[1 mark]** for correct function declaration line, including* `public`.
*Award **[1 mark]** for reducing* `WagonCount`.
*Award **[1 mark]** for return statement.*
*Award **[1 mark]** for returning the correct object.*
*Award **[1 mark]** for checking the value of the counter.*

Example:

```
public Wagon removeWagon()
{
  if (mWagonCount > 0 )
  {
    mWagonCount--;
    return mWagons[mWagonCount];
  }
  else
  {
    return null;
  }
}
```

[5 marks]

15. (a) *Award **[1 mark]** for stating an advantage and **[1 mark]** for a more complete outline, up to **[2 marks max]**.*

Example:
Supports code reuse;
Programmers can select an appropriate object class from the library and not have to design, implement and test it; ***[2 marks]***

(b) *Award **[1 mark]** for stating the difference and **[1 mark]** for describing it, for two differences, up to **[4 marks max]**.*

Example:
Programmers can be specialized;
Some programmers can develop special expertise in testing, while others focus on User Interfaces, *etc*;
Information hiding can be used to reduce module dependencies;
Different programmers can work on different classes independently because the internals of those classes are hidden;
Easier to resolve problems when many heads; ***[4 marks]***

(c) (i) String; ***[1 mark]***

(ii) int; ***[1 mark]***

(iii) Boolean; ***[1 mark]***

(d) It needs to modify the instance variable declarations;
Change the formal parameters of the constructor;
Add "get" and "set" methods; ***[3 marks]***

(e) Inheritance avoids duplicating code in the two new classes;
The new `DestinationAddress` class will have a member variable to store the special delivery instructions;
The `OriginAddress` class will have a member variable indicating where the parcel was collected from;
They will inherit common attributes from the parent;
Examples of these; ***[3 marks]***

16. (a) (i) *Award **[1 mark]** for showing the correct engines (7 and 9).*
*Award **[1 mark]** for showing the engines in the correct order.*

[0]	[1]	[2]	…
7	9		

[2 marks]

(ii) 2; ***[1 mark]***

(iii) *Award **[1 mark]** for showing the correct Wagons (23 and 214).*
*Award **[1 mark]** for showing the wagons in the correct order.*

[0]	[1]	[2]	…
23	214		

[2 marks]

(b) *Award **[1 mark]** for calling `getWeight()` on parent class.*
*Award **[1 mark]** for adding weight of individual parcels in a loop.*
*Award **[1 mark]** for using correct loop index endpoints.*
*Award **[1 mark]** for returning the correct value.*

Example:

```
public double getWeight()
{
  double totalWeight = super.getWeight();   // or totalWeight = 32000;
  for(int i=0; i < mParcelCount; i++)
  {
    totalWeight += mParcels[i].getWeight();
  }
  return totalWeight;
}
```

[4 marks]

(c) *Award **[1 mark]** for intitializing total weight to zero.*
*Award **[1 mark]** for correctly adding the weights of all the engines.*
*Award **[1 mark]** for correctly adding the weights of all the wagons.*
*Award **[1 mark]** for returning the correct value.*

Example:

```
public double getWeight()
{
  double totalWeight = 0;
  for (int i = 0; i < mEngineCount; i++ )
  {
    totalWeight += mEngines[i].getWeight();
  }
  for (int i = 0; i < mWagonCount; i++ )
  {
    totalWeight += mWagons[i].getWeight();
  }
  return totalWeight;
}
```

[4 marks]

(d) The compiler can tell which is which;
The compiler checks to see what class of object the method is being invocated on and therefore choose the correct one;
They are in different classes; ***[2 marks]***

17. (a) Modify the `Wagon` class to have a `nextWagon` member variable;
Modify the `Train` class / `Driver` class to have a `firstWagon` member variable;
Modify the `addWagon()` and `removeWagon()` methods to implement the linked list; ***[3 marks]***

(b) *Award **[1 mark]** for the correct function header.*
*Award **[1 mark]** for making the `nextWagon` point to the current `firstWagon`.*
*Award **[1 mark]** for making the `firstWagon` point to the new `Wagon`.*

Example:

```
public void addWagon(Wagon newWagon)
{
  newWagon.nextWagon = firstWagon;
  firstWagon = newWagon;
}
```

[3 marks]

(c) *Award up to **[5 marks max]**.*
The method would have to take an ID number as an argument;
It would start with the first `Wagon` stored in the `Train` class;
Check if the `Wagon` ID matched;
If so, remove that `Wagon` by altering the links;
If not, follow the link to the next `Wagon`;
It should return null if there is no `Wagon` with the ID number specified is found; ***[5 marks]***

(d) A stack; ***[1 mark]***

(e) (i) *Award **[1 mark]** for the correct function header.*
*Award **[1 mark]** for using the push method.*

Example:

```
public void addParcel(Parcel newParcel)
{
  model.push(newParcel);
}
```

[2 marks]

(ii) *Award **[1 mark]** for the correct function header.*
*Award **[1 mark]** for using the pop method.*

Example:

```
public Parcel getParcel()
{
  return model.pop();
}
```

[2 marks]

(f) *Award **[1 mark]** for stating a reason for a style convention, and a further **[1 mark]** for elaborating.*
*Award **[1 mark]** for stating a reason for a naming convention, and a further **[1 mark]** for elaborating.*

Points to consider:
Meaningful identifiers, proper indentation and adequate comments all improve the readability of code for humans and thus save money, time and effort in programming teams. ***[4 marks]***