2(a)(i)

1 mark each
- Class Horse declaration (and end where appropriate)
- All 3 attributes declared as **private** with appropriate data types (declaration or comment)
- Constructor header (and end) taking 3 parameters (constructor must be within class) …
- … constructor assigns parameters to attributes

e.g.
Python

```
class Horse:
    def __init__(self, PName, PMaxFenceHeight, PPercentageSuccess):
        self.__Name = PName #String
        self.__MaxFenceHeight = PMaxFenceHeight #Integer
        self.__PercentageSuccess = PPercentageSuccess #Integer
```

VB.NET

```
Class Horse
        Private Name As String
        Private MaxFenceHeight As Integer
        Private PercentageSuccess As Integer
   Sub New(PName, PMaxFenceHeight, PPercentageSuccess)
            Name = PName
            MaxFenceHeight = PMaxFenceHeight
            PercentageSuccess = PPercentageSuccess
        End Sub
End Class
```

2(a)(i)

Java

```
class Horse{
     private static String Name;
     private static Integer MaxFenceHeight;
     private static Integer PercentageSuccess;

     public Horse(String PName, Integer PMaxFenceHeight, Integer PPercentageSuccess){
           Name = PName;
           MaxFenceHeight = PMaxFenceHeight;
           PercentageSuccess = PPercentageSuccess;

     }
}
```

2(a)(ii)

1 mark each
- 1 get method header with no parameter …
- … returning correct attribute (without change)
- 2nd get method correct

e.g.
Python

```
def GetName(self):
   return self.__Name
def GetMaxFenceHeight(self):
   return self.__MaxFenceHeight
```

VB.NET

```
Function GetName()
    Return Name
End Function
```

2(a)(ii)

```
Function GetMaxFenceHeight()
    Return MaxFenceHeight
End Function
```

Java

```
public String GetName(){
      return Name;
}
public Integer GetMaxFenceHeight(){
      return MaxFenceHeight;
}
```

2(b)(i)

1 mark each

- Instantiating one object of type `Horse` with correct data …
- … and storing in first element of a 1D array `Horses`
- Instantiating second object of type `Horse` with correct data **and** storing in second index of the array
- Outputting name of both horse objects from array …
- … using `GetName()`

e.g.

Python

```
Horses = []
Horses.append(Horse("Beauty", 150, 72))
Horses.append(Horse("Jet", 160, 65))
print(Horses[0].GetName())
print(Horses[1].GetName())
```

VB.NET

```
Dim Horses(2) As Horse
Horses(0) = New Horse("Beauty", 150, 72)
Horses(1) = New Horse("Jet", 160, 65)
Console.WriteLine(Horses(0).GetName())
Console.WriteLine(Horses(1).GetName())
```

2(b)(i)	Java
	`Horse[] Horses = new Horse[2];`
	`Horses[0] = new Horse("Beauty", 150, 72);`
	`Horses[1] = new Horse("Jet", 160, 65);`
	`System.out.println(Horses[0].GetName());`
	`System.out.println(Horses[1].GetName());`

2(b)(ii)	1 mark for both names output:
	
	`Beauty`
	`Jet`

2(c)(i)	1 mark each
	• Class `Fence` header (and end where appropriate) with no inheritance
	• `Height` and `Risk` **private** with integer data type
	• Constructor taking 2 parameters and storing in attributes (constructor must be within class)
	• 2 get methods (no parameter) returning correct attributes (within class)
	
	e.g.
	Python
	`class Fence:`
	`  def __init__(self, PHeight, PRisk):`
	`  self.__Height = PHeight #integer`
	`  self.__Risk = PRisk #integer`
	
	`  def GetHeight(self):`
	`  return self.__Height`
	`  def GetRisk(self):`
	`  return self.__Risk`

2(c)(i)

VB.NET

```
Class Fence
        Dim Height As Integer
        Dim Risk As Integer

        Sub New(PHeight, PRisk)
            Height = PHeight
            Risk = PRisk
        End Sub
        Function GetHeight()
            Return Height
        End Function

        Function GetRisk()
            Return Risk
        End Function
End Class
```

Java

```
class Fence{
     private Integer Height;
     private Integer Risk;

     public Fence (Integer PHeight, Integer PRisk){
          Height = PHeight;
          Risk = PRisk;

     }
```

2(c)(i)

```
      public Integer GetHeight(){
            return Height;
      }
      public Integer GetRisk(){
            return Risk;
      }
}
```

2(c)(ii)

1 mark each to max 5

- Declaration/use of array `Course` of type `Fence` (with at least 4 elements)
- Taking `Height` and `Risk` as input **four** times and store/use
- Instantiating a `Fence` object for each set of valid **input** values and storing in array
- Taking each height as input **until** it is between 70 and 180 (inclusive)
- Taking each risk as input **until** it is between 1 and 5 (inclusive)

e.g.
Python

```
Course = []
for x in range(0, 4):
    Valid = False
    while Valid == False:
        Height = int(input("Enter the height in cm"))
        if(Height >= 70 and Height <= 180):
            Valid = True
    Valid = False
    while Valid == False:
        Risk = int(input("Enter the risk between 1 (easy) and 5 (hard)"))
        if(Risk >= 1 and Risk <= 5):
            Valid = True
    Course.append(Fence(Height, Risk))
```

2(c)(ii)

VB.NET

```
Dim Course(5) As Fence
Dim Height As Integer
Dim Risk As Integer
For x = 0 To 3
   Do
        Console.WriteLine("Enter the height in cm")
        Height = Console.ReadLine()
   Loop Until Height >= 70 And Height <= 180
   Do
        Console.WriteLine("Enter the risk between 1 (easy) and 5 (hard)")
        Risk = Console.ReadLine()
   Loop Until Risk >= 1 And Risk <= 5
   Course(x) = New Fence(Height, Risk)
Next
```

Java

```
Fence [] Course = new Fence [4];
for(Integer x = 0; x < 4; x++){
     do {
          System.out.println("Enter the height in cm");
          Height = Integer.parseInt(scanner.nextLine());
     } while(Height <70 || Height > 180);
     do{
          System.out.println("Enter the risk between 1 (easy) and 5 (hard)");
          Risk = Integer.parseInt(scanner.nextLine());
     }while(Risk <1 || Risk > 5);
          Course[x] = new Fence(Height, Risk);
}
```

2(d)

1 mark each

- **Method** header taking 2 parameters (and end where appropriate, returning real)
- Checking if fence height **parameter** is more than max attribute for that horse, if true multiplying percentage success by 0.2
- (Otherwise) selection checking risk value parameter between 1 and 5, multiplying modifier by percentage success
- Returning correct value as a real number in **all** instances
- Correct use of attributes and parameters throughout

e.g.

Python

```
def Success(self, Height, Risk):
    if Height > self.__MaxFenceHeight:
        return self.__PercentageSuccess * 0.2
    else:

        if Risk == 1:
            return self.__PercentageSuccess
        elif Risk == 2:
            return self.__PercentageSuccess * 0.9
        elif Risk == 3:
            return self.__PercentageSuccess * 0.8
        elif Risk == 4:
            return self.__PercentageSuccess * 0.7
        else:
            return self.__PercentageSuccess * 0.6
```

VB.NET

```
Function Success(Height, Risk)
        If Height > MaxFenceHeight Then
            Return PercentageSuccess * 0.2
        Else
```

2(d)

```
            If Risk = 1 Then
                Return PercentageSuccess
            ElseIf Risk = 2 Then
                Return PercentageSuccess * 0.9
            ElseIf Risk = 3 Then
                Return PercentageSuccess * 0.8
            ElseIf Risk = 4 Then
                Return PercentageSuccess * 0.7
            Else
                Return PercentageSuccess * 0.6
            End If
        End If

End Function
```

Java

```
public static Double Success(Integer Height, Integer Risk){
           if(Height > MaxFenceHeight){
                return Double.valueOf(PercentageSuccess) * 0.2;
           }else{
                if(Risk == 1){
                          return Double.valueOf(PercentageSuccess);
                     }else if (Risk == 2){
                          return Double.valueOf(PercentageSuccess) * 0.9;
                }else if (Risk == 3){
                          return Double.valueOf(PercentageSuccess) * 0.8;
                }else if (Risk == 4){
                          return Double.valueOf(PercentageSuccess) * 0.7;
                }else{
                          return Double.valueOf(PercentageSuccess) * 0.6;
                }
           }
}
```

2(e)(i)

1 mark each

- Calling `Success()` for **each** horse with the height and risk of all **4** fences …
- … using get methods for height and risk of each fence
- … outputting the horse name, fence number and calculated success at fence in appropriate message

e.g.

Python

```
for y in range(0, 2):
    for x in range(0, 4):
        Chance = Horses[y].Success(Course[x].GetHeight(), Course[x].GetRisk())
        print(Horses[y].GetName(), "Fence", x + 1, "chance of success is", Chance, "%")
```

VB.NET

```
    Dim Chance As Single
        For y = 0 To 1
            For x = 0 To 3
                Chance = Horses(y).Success(Course(x).GetHeight(), Course(x).GetRisk())
                Console.WriteLine(Horses(y).GetName() & " Fence " & x + 1 & " chance of
success is " & Chance & "%")
            Next
        Next
```

Java

```
     Double Chance = 0.0;
     for(Integer y = 0; y < 2; y ++){
          for(Integer x = 0; x < 4; x++){
               Chance = Horses[y].Success(Course[x].GetHeight(), Course[x].GetRisk());
               System.out.println(Horses[y].GetName() + " Fence " + (x + 1) + " chance of
success is " + Chance + "%");
          }
     }
```

2(e)(ii)	1 mark each
	• Calculating average of all 4 fences for each horse **and** outputting in suitable message
	• Identifying the highest percentage of success **and** outputting the horse's name in an appropriate message
	
	e.g.
	Python
	`AverageSuccess = []`
	`for y in range(0, 2):`
	`  Total = 0`
	`  for x in range(0, 4):`
	`  Chance = Horses[y].Success(Course[x].GetHeight(), Course[x].GetRisk())`
	`  print(Horses[y].GetName(), "Fence", x + 1, "chance of success is", Chance, "%")`
	`  Total = Total + Chance`
	`  Average = Total / 4`
	`  AverageSuccess.append(Average)`
	`  print(Horses[y].GetName(), "average success rate is", Average, "%")`
	
	`Highest = AverageSuccess[0]`
	`Winner = -1`
	`for x in range(1,2):`
	`  if Highest < AverageSuccess[x]:`
	`  Winner = x`
	`  Highest = AverageSuccess[x]`
	`print(Horses[Winner].GetName(), " has the highest average chance of success ")`

2(e)(ii)

VB.NET

```
Dim Total As Integer
Dim Chance As Single
Dim Average As Single
For y = 0 To 1
    Total = 0
    For x = 0 To 3
        Chance = Horses(y).Success(Course(x).GetHeight(), Course(x).GetRisk())
        Console.WriteLine(Horses(y).GetName() & " Fence " & x + 1 & " chance of success is
" & Chance & "%")
        Total = Total + Chance
        Average = Total / 4
        AverageSuccess(y) = Average
        Console.WriteLine(Horses(y).GetName() & " average success rate is " & Average &
"%")
    Next
Next

Dim Highest As Single
Dim Winner As Integer
Highest = AverageSuccess(0)
Winner = -1
For x = 1 To 1
    If Highest < AverageSuccess(x) Then
        Winner = x
        Highest = AverageSuccess(x)
       End If
Next x
Console.WriteLine(Horses(Winner).GetName() & " has the highest average chance of success ")
```

2(e)(ii)

Java

```
Double Total = 0.0;
Double Chance = 0.0;
Double Average = 0.0;
for(Integer y = 0; y < 2; y ++){
      Total = 0.0;
      for(Integer x = 0; x < 4; x++){
            Chance = Horses[y].Success(Course[x].GetHeight(), Course[y].GetRisk());
            System.out.println(Horses[y].GetName() + " Fence " + (x + 1) + " chance of
success is " + Chance + "%");
            Total = Total + Chance;
      }
      Average = Total / 4;
      AverageSuccess[y] = Average;
      System.out.println(Horses[y].GetName() + " average success rate is " + Average +
"%");
}
Double Highest = AverageSuccess[0];
Integer Winner = 0;
for(Integer x = 1; x < 2; x++){
      if(Highest < AverageSuccess[x]){
            Winner = x;
            Highest = AverageSuccess[x];
      }
}
System.out.println(Horses[Winner].GetName() + " has the highest average chance of
success");
```

2(e)(iii)

1 mark each

- Outputting showing correct input values for all fences, **and** correct chance for each horse on each jump
- Outputs of average chance of each horse **and** horse name with highest average

e.g.

```
Enter the height in cm152
Enter the risk between 1 (easy) and 5 (hard)5
Enter the height in cm121
Enter the risk between 1 (easy) and 5 (hard)1
Enter the height in cm130
Enter the risk between 1 (easy) and 5 (hard)3
Enter the height in cm145
Enter the risk between 1 (easy) and 5 (hard)4
Beauty Jump 1 chance of success is 14.4 %
Beauty Jump 2 chance of success is 72 %
Beauty Jump 3 chance of success is 57.6 %
Beauty Jump 4 chance of success is 50.4 %
Beauty average success rate is 48.6 %
Jet Jump 1 chance of success is 39.0 %
Jet Jump 2 chance of success is 65 %
Jet Jump 3 chance of success is 52.0 %
Jet Jump 4 chance of success is 45.5 %
Jet average success rate is 50.375 %
Jet has the best chance of winning
```