

Dear Candidate – This programming exercise is designed to test your understanding of basic Scala concepts and problem solving techniques.

Acceptance Criteria:

1. Please submit your exercise using GitHub or other public git repository. Include a short readme with instructions on how to run your program.
2. We recommend attempting the problem in Scala even if you are not familiar with the language. However we will accept the solution in another language, but you may have to transform the sample data to an appropriate data structure.
3. Be prepared to speak about your design decisions during your on-site interview.

Problem 1

Background: The TST cruise application receives pricing and rate information from a third party data provider. We make two calls to this provider to receive a list of rates and a list of cabin prices. We can use this data to solve several problems for our customers. The problem we'll be focusing on for this exercise will be finding the best price for a particular rate group.

Cabin Price: The price for a specific cabin on a specific cruise. All cabin prices will have a single rate attached.

Rate: A rate is a way to group related prices together. A rate is defined by its Rate Code and which Rate Group it belongs to. For example. (MilAB, Military) and (Sen123, Senior)

Rate Group: Specific rates are grouped into a related rate group. There is a one-to-many relationship between rate groups and rates (A rate group is made up of many rates, but a rate can only belong to a single rate group) Some examples of rate groups are: Standard, Military, Senior, and Promotion.

1. Write a function that will take a list of rates and a list of prices and returns the best price for each rate group. We've written the function and case class definitions below for you to use.

```
def getBestGroupPrices(rates: Seq[Rate],
                      prices: Seq[CabinPrice]): Seq[BestGroupPrice] =
  ???

case class Rate(rateCode: String, rateGroup: String)

case class CabinPrice(cabinCode: String,
                     rateCode: String,
                     price: BigDecimal)

case class BestGroupPrice(cabinCode: String,
                         rateCode: String,
                         price: BigDecimal,
                         rateGroup: String)
```

2. On startup, your program should run the following sample data through your function and output the sequence of BestGroupPrices. We included the expected output below.

Input - Rates:

```
Rate(M1, Military)
Rate(M2, Military)
Rate(S1, Senior)
Rate(S2, Senior)
```

Input - Cabin Prices:

```
CabinPrice(CA, M1, 200.00)
CabinPrice(CA, M2, 250.00)
CabinPrice(CA, S1, 225.00)
CabinPrice(CA, S2, 260.00)
CabinPrice(CB, M1, 230.00)
CabinPrice(CB, M2, 260.00)
CabinPrice(CB, S1, 245.00)
CabinPrice(CB, S2, 270.00)
```

Expected Output - Best Cabin Prices:

```
BestGroupPrice(CA, M1, 200.00, Military)
BestGroupPrice(CA, S1, 225.00, Senior)
BestGroupPrice(CB, M1, 230.00, Military)
BestGroupPrice(CB, S1, 245.00, Senior)
```

Problem 2

Background: Cruise bookings can have one or more Promotions applied to them. But sometimes a Promotion cannot be combined with another Promotion. Our application has to find out all possible Promotion Combinations that can be applied together.

1. Implement a function to find all PromotionCombos with maximum number of combinable promotions in each. The function and case class definitions are supplied below to get you started.

```
case class Promotion(code: String, notCombinableWith: Seq[String])
case class PromotionCombo(promotionCodes: Seq[String])
```

```
def allCombinablePromotions(allPromotions: Seq[Promotion]): Seq[PromotionCombo]
= ???
```

2. Implement a function to find all PromotionCombos for a given Promotion from given list of Promotions. The function definition is provided.

```
def combinablePromotions(  
    promotionCode: String,  
    allPromotions: Seq[Promotion]): Seq[PromotionCombo] = ???
```

3. On startup your program should run through the following sample data and output the sequence of PromotionCombos.

Input - Promotions:

```
Promotion(P1, Seq(P3))      // P1 is not combinable with P3  
Promotion(P2, Seq(P4, P5))  // P2 is not combinable with P4 and P5  
Promotion(P3, Seq(P1))      // P3 is not combinable with P1  
Promotion(P4, Seq(P2))      // P4 is not combinable with P2  
Promotion(P5, Seq(P2))      // P5 is not combinable with P2
```

Expected Output for All Promotion Combinations:

```
Seq(  
    PromotionCombo(Seq(P1, P2)),  
    PromotionCombo(Seq(P1, P4, P5)),  
    PromotionCombo(Seq(P2, P3)),  
    PromotionCombo(Seq(P3, P4, P5))  
)
```

Expected Output for Promotion Combinations for promotionCode="P1":

```
Seq(  
    PromotionCombo(Seq(P1, P2)),  
    PromotionCombo(Seq(P1, P4, P5))  
)
```

Expected Output for Promotion Combinations for promotionCode="P3":

```
Seq(  
    PromotionCombo(Seq(P3, P2)),  
    PromotionCombo(Seq(P3, P4, P5))  
)
```