8

Description

Your task here is to implement a **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider **default visibility** of classes, data fields and methods unless mentioned otherwise.

Specifications

```
class definitions:
   class TransactionParty:
      Variables:
         String seller
         String buyer
      Method definitions:
         Implement a parameterized constructor
         TransactionParty(String seller, String buyer):
          Visibility: public

  class Receipt:
      Variables:
         TransactionParty transactionParty
         String productsQR
     Method definitions:
         Implement a parameterized constructor
          Receipt(TransactionParty transactionParty, String
productsQR):
          Visibility: public

  class GenerateReceipt:
       method definitions:
         verifyParty(Receipt r):
            Return type: int
            Visibility: public

         calcGST(Receipt r):
            Return type: String
            Visibility: public
```

Tasks

Class **TransactionParty**

- define the **String** variable **seller**

- define the **String** variable **buyer**

-implement a parameterized **constructor**

class Receipt

-declare the below instance variables and implement a parameterized constructor to initialize them.

1. **TransactionParty transactionParty**

2. **String productsQR**

- a string of the format: "**<Rate>,<Quantity>@<Rate>,<Quantity>@<Rate>,<Quantity>**"
- **e.g:** "250,10@100,3@50,7"
- only 3 products' **Rate** and **Quantity** present in the string

To access a variable in **TransactionParty** class through **Receipt** object we use:

```
<Receipt(obj)>.<Receipt(variable)>.<TransactionParty(variable)>
```

e.g: To access "**seller**" name from the **Receipt** object **r** we use:

```
r.transactionParty.seller;
```

class **GenerateReceipt**

-Implement the below methods for this class:

1.int verifyParty(Receipt r):

In this method, you have to use regex to check if the names
- **buyer** and **seller** of **TransactionParty** object available in **Receipt r**, are valid or not. Validate the names as per the below condition:

Names:

- should start and end with alphabets (uppercase/lowercase)
- can have white-spaces in between alphabets
- can have a single quote symbol OR a hyphen symbol in between alphabets
- **e.g: Daniel D'Cruz, Giselle Dawn-Wright** and **Giselle Dawn** are valid names.

Return:

- 2 if the both names are valid
- 1 if only one name is valid
- 0 if both names are invalid.

2. String calcGST(Receipt r):

In this method, you have to use the **productsQR** variable of the **Receipt r** object to extract the **Quantity** and **Rate** of 3 products.

- The variable **productsQR** will have a string value of the format **<Rate>,<Quantity>@<Rate>,<Quantity>@<Rate>,<Quantity>**
- **Rate** is the price rate of the product
- **Quantity** is the number of units bought
- **e.g**: **productsQR** = "**250,10@100,3@50,7**" has the **Rate** as 250, 100 and 50 while **Quantity** as 10, 3 and 7 respectively.

Use the **GST_Rate** as **12%.**

Calculate the value of GST using the formula:

```
GST = (Rate1 * Quantity1 + Rate2 * Quantity2 + Rate3 * Quantity3) *
GST_Rate
```

The data type of GST should be **int** only. (Do NOT round-off the result)

Return the value of **GST** as a string value, using the **toString()** method.

NOTE

- You can make suitable function calls and use **RUN CODE** button to check your **main()** method output.

7

Description

Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

Specifications:

```
class definitions:
   class Header:
               Variables:
                       String from
                       String to
               Implement a parameterized constructor to initialize all
the instance variables.

   class Email:
               Variables:
                       Header header
                       String body
                       String greetings
```

```
                Implement a parameterized constructor to initialize all
the instance variables.
  class EmailOperations:
                Methods:
                       emailVerify(Email e): Use regular expression to
verify if the two email-ids in the Header class is valid or not.[Return
type explained in Task part].
                              Return type:int
                              Visibility: public

                       bodyEncryption(Email e): Use Ceasar cipher(Shift-
3) to encrypt the body of the email.[To know more refer the Task part]
                              Return type:String
                              Visibility: public

                       greetingMessage(Email e): In this method you have
to return a greeting messgae. The greet part should be taken from
greetings variable and signature(name) should be taken from Header's
'from' email address.[To know more refer the Task part]
                              Return type:String
                              Visibility: public
```

Class Variables:

- **class Header:** It contains two email id 'from' and 'to'. 'from' signifies the sender's email address and 'to' signifies receiver's email address.
- **class Email:** This class contains three parts: first Header header which has two email address from and to,the second body which contains the message to send and third greetings which contains greetings such as "Regards", "Thank you", etc.

To access a variable in Header class through Email object we use:

```
<Email(obj)>.<Email(variable)>.<Header(variable)>
```

Example to access "from" address from the Email object e we use : e.header.from;

Tasks:

- Implement the two classes Email and Header class according to the specifications.
- Implement the three methods in the EmailOperations class:

1. emailVerify (Email e)
2. bodyEncryption (Email e)
3. greetingMessage (Email e)

Method Description:

1. emailVerify(Email e):

- In this method you have to use regex to check if the email-address to and from in Header class is valid or not. Validation is based on:
- Email address should start with alphabets(capital/small) or _(underscore).
- Email address should have only one @ followed by alphabets.
- Email address should end with .(dot) followed by alphabets.
- e.g: amit@doselect.com, _ami@doselect.in are valid addresses, but 1ami@dos.com, amit@doselect are invalid addresses.
- Return 2 if the both email addresses are valid return 1 if one is valid, and 0 if both are invalid.

2. bodyEncryption(Email e):

- In this method, you have to use Caesar cipher(shift of 3) to encrypt the body part of the Email return the encrypted string.
- Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. Here the number of shift is 3.
- e.g: str = "Hi There Hows you", after encryption becomes "Kl Wkhuh Krzv brx". H get converted to K that is a shift of 3 alphabets ahead.
- Letters which are capital should be capital and small should be small in Encrypted message. Take care of the spaces.

3. greetingMessage(Email e):

- In this method, you have to return a concatenated string which contains the greetings variable from Email class and Name of the person who is sending the mail(from variable in the Header class).
- The name part should not contain anything which is after @ in the email id.
- e.g: if greetings = "Regards" and from = "Amit@doselect.com" then you have to return the message "Regards Amit"

Important:

- To check your program you have to use the main() function(in Source class) given in the stub. You can make suitable function calls and use **RUN CODE** button to check your **main()** function output.

Your task here is to implement a **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

Specifications:

```
class Candidate
  data members:
   name: String
   id: int
   age: int
   gender: String
   department: String
   yearOfJoining: int
   salary: double
    visibility: private

  Candidate(int id, String name, int age, String gender, String
department, int yearOfJoining, double salary): constructor with public
visibility
  Define getter setters with public visibility
  toString() method has been implemented for you

class Implementation:
 method definition:
  getCount(List<Candidate> list):
   return type: Map<String, Long>
   visibility: public

  getAverageAge(List<Candidate> list):
   return type: Map<String, Double>
   visibility: public

  countCandidatesDepartmentWise(List<Candidate> list):
   return type: Map<String, Long>
   visibility: public

  getYoungestCandidateDetails(List<Candidate> list)
   return type: Optional<Candidate>
   visibility: public
```

Task:

class Candidate:

- define the **String** variable **name**

- define the **int** variable **id**

- define the **int** variable **age**

- define the **String** variable **gender**

- define the **String** variable **department**

- define the **int** variable **yearOfJoining**

- define the **double** variable **salary**

-define a **constructor** and **getter setters** according to the above specifications

-**toString()** method has been implemented for you as a part of the code stub

class Implementation:

Implement the below method for this class using in Stream API:

- static Map<String, Long> getCount(List<Candidate> list): get the count of male and female employees from the list, put it into a Map and return the Map
- static Map<String, Double> getAverageAge(List<Candidate> list): return the average age of male and female employees
- static Map<String, Long> countCandidatesDepartmentWise(List<Candidate> list): **count and return** the number of employees in each department
- static Optional<Candidate> getYoungestCandidateDetails(List<Candidate> list): Get the details of youngest male employee in the product development department

Sample Input

```
 List<Candidate> list = new ArrayList<>();
  list.add(new Candidate(111, "Jiya Brein", 32, "Female", "HR", 2011,
25000.0));
  list.add(new Candidate(144, "Scarlet Jhonson", 28, "Male", "Product
Development", 2014, 32500.0));
------------------------------------------------
list //Input for all the methdos
```

Sample Output

```
{Male=1, Female=1}
--------------------------------
{Male=28.0, Female=32.0}
--------------------------------
{Product Development=1, HR=1}
--------------------------------
Optional[Employee [id=144, name=Scarlet Jhonson, age=28, gender=Male,
department=Product Development, yearOfJoining=2014, salary=32500.0]]
```

NOTE

You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.

5

Your task here is to implement a Java code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

Specifications:

```
class definitions:
  class Employee:
    data fields:
      name: String
      salary: int
    Implement a Constructor using the class variables.
    Implement getter setter methods with public visibility.
  class EmployeeInfo:
    enum definition:
      named constants: BYNAME
                       BYSALARY
    method definitions:
      sort(List<Employee> emps, final SortMethod method): Method to
return sorted list by name and by salary using SortMethod
           Return type: List<Employee>
           Visibility: public
      isCharacterPresentInAllNames(Collection<Employee> entities,
String character): method to check if Employee list contains a name
starting with a specific character
           Return type: boolean
           Visibility: public
```

Task:

Create an Employee class which has the following members:

```
 String name;
 int salary;
```

- Define parameterized ***constructor***.
- Define ***getter*** method for all instance variables with public visibility.(getName(),...)

- Define ***setter*** methods for all instance variables with public visibility.(setName(),....)

Create an **EmployeeInfo** class which performs following operations (as per the given requirements) using **StreamAPI**:

- **enum SortMethod** : representing a group of named constants **BYNAME** and **BYSALARY**
- **sort(List<Employee> emps, final SortMethod method)**: Method to return sorted list by **name** and by **salary** using **SortMethod**
- **isCharacterPresentInAllNames(Collection<Employee> entities, String character):** Method to check if Employee list contains a name starting with a specific character

Implement using **Lambda expressions**.

Following has been done for you:

- **Main()** method containing list of **Employees**
- **String toString()** method, it's part of code stub, don't edit it else your ***test-cases might fail***

Sample Input

```
List<Employee> emps = new ArrayList<>();
emps.add(new Employee("Mickey", 100000));
emps.add(new Employee("Timmy", 50000));
emps.add(new Employee("Annny", 40000));
```

Sample Output

```
[<name: Annny salary: 40000>, <name: Mickey salary: 100000>, <name:
Timmy salary: 50000>]
[<name: Annny salary: 40000>, <name: Timmy salary: 50000>, <name:
Mickey salary: 100000>]
false
```

NOTE

- Do not use any **for** loops or other control structures.
- Use the stream API methods for your implementations, else the test-cases might fail.
- You CAN implement the **main()** method to check the implementation of your methods in the solution.

- Upon implementation of **main()** method, you can use the **RUN CODE** button to pass input data in the method calls and arrive at some output.

4

Description

Case Study:

You have to create business logic that simulates a job agency trying to search for openings at a company. The company has the following requirements from the candidates for their offerings:

- The candidate must be atleast 19 years of age. Otherwise, a NotEligibleException needs to be thrown with message "You are underage for any job"
- If the candidate is atleast 21 and the highest qualification is a B.E, then he/she is eligible for the role of a junior developer. In this case, a string needs to be returned as "We have openings for junior developer"
- The candidate is atleast 26 years of age and the highest qualification is an M.S or a PhD, then he/she is eligible for the role of a senior developer. In this case, a string needs to be returned as "We have openings for senior developer"
- If the candidate is atleast 19 years of age and the highest qualification is not any of B.E, M.S or PhD, then an exception named NotEligibleException needs to be thrown with the message "We do not have any job that matches your qualifications"
- For all other cases, a string needs to be returned as "Sorry we have no openings for now"
- You are supposed to create a class called CompanyJobRepository, which has a static method getJobPrediction() to meet the above requirements.

Your task here is to implement a **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

```
class definitions:
class CompanyJobRepository :
  method definitions :
    static getJobPrediction(int age, String highestQualification):
      return type: String

class Source :
 visibility: public
  method definitions :
    searchForJob(int age, String highestQualification):
       return type: String
       visibility: public
```

Task

On the basis of above case study implement the below classes and methods:

class **CompanyJobRepository**

-Implement the below methods for **CompanyJobRepository**

-static String getJobPrediction(int age, String highestQualification):

- Refer the case study above for the business logic

class **Source**

-Implement the below methods for Source

-String searchForJob(int age, String highestQualification):

- if age >= 200 or age <= 0, throw NotEligibleException with the message "The age entered is not typical for a human being"
- Otherwise, get the job predictions from CompanyJobRepository.
- You have to handle the NotEligibleException thrown by getJobPrediction(), in which case you have to return the message of the exception caught.

Class **NotEligibleException**

- Define custom exception class NotEligibleException by **extending** the **Exception** class.
- Define a parameterized constructor with a String argument to pass the message to the super class.

NOTE

- You can make suitable function calls and use **RUN CODE** button to check your **main()** method output.

3

Description

In **Java**, we can use more than one catch block with the try block. Generally, multiple catch block is used to handle different types of exceptions, which means each catch block is used to handle different types of exceptions.

If you use multiple catch blocks for the same type of exception, then it will give you a compile-time error because **Java does not allow you to use multiple catch block for the same type of exception**. A catch block is always preceded by the try block.

Write a program to demonstrate Multiple Exceptions.

Specifications:

```
class Activity:
    data fields:
      String string1
      String string2
      String operator
    Constructor to initialize the class variables.

class Source:
              method definitions:
                     handleException(Activity a): implement try-catch
blocks and throw different exceptions as described under Tasks
                            return type: String
                            visibility: public

                     doOperation(Activity a): implement switch
statement to calculate Result based on value of Operator
                            return type: String
                            visibility: public
```

You have to implement the following methods under Source class:

- **handleException (Activity a)** - In this function you have to check for exceptions.
- **doOperation (Activity a)** - this function should implement the string operation between **string1** and **string2** for the operator **operator**.
- If **operator** = '+', concat the strings **string1** and **string2**.
- **e.g.** for **string1 = "hello"** and **string2 = "world"**, then **result** = **"helloworld"**
- If **operator** = '-', replace the contents of **string2** in **string1** with empty string.
- **e.g.** If **string1 = "helloworld"** and **string2 = "world"**, then **result = "hello"**

Tasks:

In the function **handleException (Activity a)**:

- Check that the value of either **string1** or **string2** variable is **null**, then throw appropriate exception for **NullPointerException** and return "**Null values found**".
- Check if the value of **operator** variable is not equal to these string operators ((+ or -) using logical AND operator. If the condition is true then throw and return the default exception with the Operator as the return message.
- If no exception is found return "**No Exception Found**".

In the function **doOperation (Activity a)**:

- perform the string operations, using switch statement and return the correct value.

IMPORTANT:

- If you want to test your program, you can implement a **main()** function given in the stub and you can use **RUN CODE** to test your main() provided you have made valid function calls with valid data required.

Description

Your task here is to implement **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider **default visibility** of classes, data fields, and methods unless mentioned.

Specifications

```
class definitions:
 class Mobile:
   data member:
      HashMap<String, ArrayList<String>> mobiles = new HashMap<>()
   method definition:
      addMobile(String company, String model)
         return type: String
         visibility: public

      getModels(String company)
         return type: ArrayList<String>
         visibility: public

      buyMobile(String company, String model)
         return type: String
         visibility: public
```

Task

Class Mobile

-define the object of HashMap<String, ArrayList<String>> with variable name **mobiles**.

- The **String** defines the **name of the company** and the **Arraylist<String>** will have list of models.

Implement the below methods for this class:

-String addMobile(String company, String model):

- Write a code to add **a company** and its **model** in **mobiles** map as given below
- If the **company** does not exist in the map already, add the **company** and its **model** into the map. (Note: Add **model** into a new **ArrayList<String>** and add this list into map as value)
- If the **company** already exist in the map, append the given **model** into the corresponding model list.
- Return "**model successfully added**" after performing the above operations

-ArrayList<String> getModel(String company):

- Write a code to get the Model list for the given company from Map **mobiles**.

- Return **null** if the given company doesn't exist or doesn't have any model, else return the **List<String>** of all the models.

-String buyMobile(String company, String model):

- Write a code to buy a mobile.
- Remove the mobile **model** from the list according to the **compnay** and **model** given. In case there are two same models then remove one and return the message "mobile sold successfully"
- Return a message "**item not available" i**f the **company** or corresponding **model** is not present in the Map

Sample Input

```
Mobile obj = new Mobile();
obj.addMobile("Oppo", "K3");
obj.getModels("Oppo");
obj.buyMobile("Oppo", "K3");
```

Sample Output

```
model successfully added
[K3]
mobile sold successfully
```

NOTE:

- You can make suitable function calls and use **RUN CODE** button to check your **main()** method output.

1

escription

Your task here is to implement a **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

Specifications:

```
class definitions:
 class ArrayListOps:
 method definitions:
  makeArrayListInt(int n): Method to create an arrayList with same
number of elements as n and set elements to 0
      return type: ArrayList<Integer>
      visibilty: public
  reverseList(ArrayList<Integer> list): Method to Reverse list
      return type: ArrayList<Integer>
      visibilty: public

  changeList(ArrayList<Integer> list, int m, int n): Method to change
all occurences of m to n in list
      return type: ArrayList<Integer>
      visibilty: public
```

Task:

Your task is to create a class **ArrayListOps** and implement the following:

1. makeArrayListInt(int n): Method to create an A**rrayList** with number of elements as n and ***set** elements* to **0**.

- If number of elements **n** is 4 , then the method should return a list containing **[0,0,0,0]**

2. reverseList(ArrayList<Integer> list): Method to ***Reverse*** **list**

3. changeList(ArrayList<Integer> list, int m, int n): Method to change all **occurences** of **m** to **n** in **list**

Important:

- To check your program, you can use the **main()** method (in Source class) given in the stub. You can make suitable function calls and use **RUN CODE** button to check your main() function output.

Sample Input

```
ArrayList<Integer> list = new ArrayList<Integer>(Arrays.asList(10, 25,
33, 28, 10, 12));
n = 4(method makeArrayListInt)
m = 28, n = 20(method changeList)
```

Sample Output

```
[0, 0, 0, 0]
[12, 10, 28, 33, 25, 10]
[12, 10, 20, 33, 25, 10]
```

NOTE:

- The above **Sample Output** is only for demonstration purposes and will be obtained if you implement the **main()** method with all method calls accordingly.
- Upon implementation of **main()** method, you can use the **RUN CODE** button to pass input data in the method calls and arrive at the **Sample Output**.