Q1

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:**

enum definition:

enum FoodType:

VEG, NONVEG

class definitions:

class Consumer:

final String name

final int age

final FoodType foodType

visibility : private

Define a parameterized constructor **with** **public** visibility

Implment getters **with** **public** visibility

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

**class** Implementation:

getNonVegetarianList(**List**<Consumer> consumer):

**return** **type**: **List**<Consumer>

visibility: **public**

sortConsumerByAge(**List**<Consumer> consumer):

**return** **type**: **List**<Consumer>

visibility: **public**

**Task:**

**enum :**has been defined for you in the code stub

**class Consumer:**

- define the data members according to above specifications

**-**define a **constructor** and **getters**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:**

* **List<Consumer> getNonVegetarianList(List<Consumer> consumer):**

fetch the details where FoodType is NONVEG, put into a list and return the list

* **List<Consumer> sortConsumerByAge(List<Consumer> consumer):**

sort the list of consumers by age and return it(in ascending order)

*Refer Sample Input Output for more details*

**Sample Input**

Implementation imp = **new** Implementation();

 Consumer p = **new** Consumer("Sarah", 45, FoodType.VEG);

 Consumer p1 = **new** Consumer("John", 26, FoodType.NONVEG);

 Consumer p2 = **new** Consumer("Mirra", 7, FoodType.NONVEG);

**List**<Consumer> consumers = Arrays.asList(p, p1, p2);

imp.getNonVegetarianList(consumers)

imp.sortConsumersByAge(consumers)

**Sample Output**

[Consumer{name='John', age=26, foodType=NONVEG}, Consumer{name='Mirra', age=7, foodType=NONVEG}]

---------------------METHOD 1------------------------------

[Consumer{name='Mirra', age=7, foodType=NONVEG}, Consumer{name='John', age=26, foodType=NONVEG}, Consumer{name='Sarah', age=45, foodType=VEG}]

---------------------METHOD 2------------------------------

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

**TAGS**

* Stream API
* Lambda

Q2

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Material:

int ﻿mass

int temperature

visibility : public

Material(int mass, int temperature) : Constructor **with** **public** visibility

﻿method definitions:

﻿flowOfHeat() throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿ checkMaterial() throws **Exception**:

**return** **type** : **String**

visibility : **public**

﻿

**class** MaterialException extends **Exception**:

﻿method definitions:

﻿ MaterialException(**String** msg)

visibility: **public** ﻿

**Task**

**Class** **Material**

**-define all the data members as per the given specifications.**

**-define the constructor with public visibility.**

**-Implement the below methods for this class:**

**-String** **checkMaterial() throws Exception:**

* Write a code to check whether the material is valid or not.
* **throw a MaterialException** if the mass is negative with a message "**Mass cannot be negative**".
* **throw a MaterialException** if the mass is zero with a message "**Mass cannot be zero**"
* if no exception is found return "**Valid material"**

**-String** **flowOfHeat() throws Exception:**

* If **checkMaterial()** method throws a **MaterialException** then returns a message "**Invalid Material**".(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* **If no exception is found then return "Both in equilibrium".**

**Sample Input**

Material m=**new** Material(12,32);

String s=m.flowOfHeat();

**Sample Output**

**Both in equilibrium**

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

**TAGS**

* Exceptions
* Exceptions and Exception Handling

Q3

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Problem:

data members:

int credits

String type

Problem(int credits, String type): constructor **with** **public** visibility

**class** Checker:

method definitions:

checkProblem(Problem p) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

approveProblem(Problem p):

**return** **type**: **String**

visibility: **public**

﻿

**class** ProblemException:

method definitions:

ProblemException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Problem**

**-**define the **int**variable **credits and string variable type.**

**-**define a **constructor** according to the above specifications.

Class **Checker**

**Implement the below methods for this class:**

**-String** **checkProblem(Problem p) throws Exception:**

* Write a code to validate the criteria for approving the problem.
* **throw a ProblemException**if credits of the problem are less than 10 with a message "**Insufficient credits".**
* **throw a ProblemException**if the type of the problem is **"String"** with a message "**String problem found".**
* **throw a ProblemException**if the type of the problem is **"Generic"** with a message "**Generic problem found".**
* **throw a ProblemException**if the type of the problem is **"I/O"** with a message "**I/O problem found".**
* else return "**Exception problem found**".

**-String approveProblem(Problem p):**

* Write a code that approves the problem.
* If **checkProblem**method throws a **ProblemException**then returns a message "**Not approved**".(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* If no exception is found then return a message "**Problem approved**".

**Sample Input**

Problem p=**new** Problem(41,"Exception");

Checker c=**new** Checker();

String s = c.checkProblem(p);

String t=c.approveProblem(p);

**Sample Output**

exception problem found

problem approved

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

**TAGS**

* Exceptions
* Exceptional Handling

Q4

**DESCRIPTION**

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Rating:

data members:

int imdbRating

int nominee

Rating(int imdbRating, int nominee): constructor **with** **public** visibility

**class** Validator:

method definitions:

canBeConsideredForTheAward(Rating rating) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

sendInvite(Rating rating) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** MovieRatingException:

method definitions:

MovieRatingException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Rating**

**-**define the **int**variable **imdbRating.**

**-**define the **int** variable **nominee**

**-**define a **constructor** according to the above specifications.

Class **Validator**

**Implement the below methods for this class:**

**-String** **canBeConsideredForTheAward(Rating rating) throws Exception:**

* Write a code to validate the criteria for getting the award.
* **throw a MovieRatingException**if **imdbRating** is less than **7**with the message "**Movie not eligible for Filmfare award**".
* **throw a MovieRatingException**if **nominee** is less than **4** with the message "**Minimum 4 nominee required**".
* If no above exception found then return a string message "**Considered for the award**".

**-String sendInvite(Rating rating):**

* Write a code to send an invite to the nominee.
* If **canBeConsideredForTheAward** method throws a **MovieRatingException** then return a message "**Not invited**".(Use try-catch block)
* If it throws any other exception then return a message "**other exception**".
* If no exception found then return a message "**Actors and Directors Invited**".

**Sample Input**

Rating rating = **new** Rating(9, 7);

Validator v = **new** Validator();

------------------------------------------------------

String s = v.canBeConsideredForTheAward(rating);

String t = v.sendInvite(rating);

s.toLowerCase();

t.toLowerCase();

**Sample Output**

considered **for** the award

actors **and** directors invited

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

**TAGS**

* Exception Handling

Q5

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Validator:

method definitions:

checkComment(String **comment**) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

commentTheString(**String** **comment**) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** SpamCommentException:

method definitions:

SpamCommentException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Validator**

**Implement the below methods for this class:**

**-String** **checkComment(String comment):**

* Write a code to validate the comment.
* throw a **SpamCommentException,** if comment has these words["**abcde**", "**lmno**", "**pqrst**", "**wxyz**"] in it, with the message "**spam comment**".
* throw a **SpamCommentException**, if a comment contains more than 2 words from the above list, with the message "**account ban due to spam comment**". Note same words with a frequency of more than 2 will come in this category.
* return a string message "**comment is not spam**" If none of the above exceptions is found.

*Refer the example for better understanding.*

*s0 = "hello my name is steve and using****abcde****"*

*s1 = "hello my name is steve and using****abcde abcde****"*

*s2 = "hello my name is steve and using****abcde lmno pqrst****"*

*s3 = "hello my name is steve and using****abcde abcde lmno****"*

*s0 and s1 come under****spam comment****message.*

*s2 and s3 comes under the****account ban****message.*

**-String commentTheString(String comment):**

* Write a code to put the comment on the post.
* If a checkComment method throws a **SpamCommentException** ,then return a message of that exception(Use try-catch block).
* If it throws any other exception then return a message "**other exception**".
* If no exception is found then return a message "**comment posted**".

class **SpamCommentException**

-Define **SpamCommentException** class derived from Exception class

**Sample Input**

Validator obj = new Validator();

-------------------------------------------

obj.checkComment("hello my name is steve");

obj.commentTheString("my **comment** **is** **safe** **to** post");

**Sample Output**

**comment** **is** **not** spam

**comment** posted

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

**TAGS**

* Java
* Exception Handling in Java

Q6

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.

**Specifications**

class definitions:

﻿class Credentials:

﻿ data members:

userName: String

password: String

Credentials(String userName, String password): constructor **with** **public** visibility

**class** Implementation:

﻿ method definitions:

﻿ passwordValidator(Credentials details) throws **Exception**

﻿ **return** **type**: **String**

visibility: **public**

signUp(Credentials details)

**return** **type**: **String**

﻿ visibility: **public**

**class** MissingCharacterException:

﻿method definitions:

﻿ MissingCharacterException(**String** **str**)

visibility: **public**

﻿

**class** LengthMismatchException:

﻿method definitions:

﻿ LengthMismatchException(**String** **str**)

visibility: **public**

**Task**

**Class** **Credentials**

- define the **String** variable **userName**.

**-**define the **String**variable **password**.

-Define a parameterized constructor as specified above.

**Class** **Implementation**

**Implement the below methods for this class:**

**-String passwordValidator(Credentials details):**

* Write a code to validate the password.
* If the password doesn't contain at least one special character, one Upper case alphabet, one lower case alphabet, and one digit then **throw a MissingCharacterException** with a message "**password must contain at least one special character, at least one upper case alphabet, at least one lower case alphabet, and at least one digit**".
* If the password length is not between 8 to 40(inclusive) then **throw a LengthMismatchException** with a message "**length of the password is not between 8 to 40**".
* If no exception found then return a message "**valid password**".

**-String signup(Credentials details):**

* Write a code to sign up using the given details.
* Use try-catch to validate the password using **passwordValidator** method, If the password is valid then return "**signup successfully**".
* If **MissingCharacterException** or **LengthMismatchException** found then return "**invalid password**" in the catch block.
* If any other exception found then return "**other exception**" in the catch block.

**Class** **MissingCharacterException**

* define custom exception class **MissingCharacterException** by **extending** the **Exception** class.
* define a parameterized constructor with a String argument to pass the message to the super class.

**Class** **LengthMismatchException**

* define custom exception class **LengthMismatchException** by **extending** the **Exception** class.
* define a parameterized constructor with a String argument to pass the message to the super class.

**Sample Input**

Implementation obj = **new** Implementation();

-------------------------------------------------------

obj.passwordValidator(**new** Credentials("Steve", "Akjsdhhj@1"));

obj.signUp(**new** Credentials("Bob", "jsdhJS12\*&$1"));

Sample Output

valid password

signup successfully

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q7

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.

**Specifications**

class definitions:

﻿class Person:

﻿ data members:

name: String

dateOfBirth: String

email: String

Person(String name, String dateOfBirth, String email): constructor **with** **public** visibility

**class** Implementation:

﻿ method definitions:

﻿ validator(Person details) throws **Exception**

﻿ **return** **type**: **String**

visibility: **public**

submitDetails(Person details)

**return** **type**: **String**

﻿ visibility: **public**

**class** InvalidDateException:

﻿method definitions:

﻿ InvalidDateException(**String** **str**)

visibility: **public**

﻿

**class** InvalidEmailException:

﻿method definitions:

﻿ InvalidEmailException(**String** **str**)

visibility: **public**

**Task**

**Class** **Person**

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

**-**define the **String**variable **dateOfBirth**.

**-**define the **String**variable **email**.

-Define a parameterized constructor as specified above.

**Class** **Implementation**

**Implement the below methods for this class:**

**-String validator(Person details):**

* Write a code to validate the details of the Person.
* **Date of birth**(dd-mm-yyyy) is present in the given format,  **throw an InvalidDateException** with the message "**date year must be less than 2000**" if the date and year are not less than 2000
* **throw an InvalidEmailException** with the message "**only @doselect.com domain email are accepted**", if the email is not of '**@doselect.com**' domain
* return String "**valid details**", if none of the above exceptions is found

**-String submitDetails(Person details):**

* Write a code to submit the details.
* Use try-catch to validate the Person details using **the validator(Person details)** method. If the details are valid then return "**details submitted successfully**".
* If **InvalidDateException** or **InvalidEmailException** is found then return "**invalid details**" in the catch block.
* If any other exception is found then return "**other exception**" in the catch block.

**Class** **InvalidDateException**

* define custom exception class **InvalidDateException** by **extending** the **Exception** class.
* define a parameterized constructor with a String argument to pass the message to the super class.

**Class** **InvalidEmailException**

* define custom exception class **InvalidEmailException** by **extending** the **Exception** class.
* define a parameterized constructor with a String argument to pass the message to the super class.

**Sample Input**

Person data = **new** Person("Steve", "12-02-1998", "Steve12@doselect.com");

Implementation obj = **new** Implementation();

-----------------------------------------------

obj.validator(data);

obj.submitDetails(data);

**Sample Output**

valid details

details submit successfully

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q8

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 are public unless mentioned otherwise.

**Specifications**

class definitions:

﻿class Product:

Data members:

﻿﻿ name: String

﻿ price: double

﻿ coupon: String

Define a parameterized constructor for all the data member.

class Validator:

﻿ validateCoupon(Product p)throws Exception

﻿return type: String

visibility: public

netPrice(Product p)

type: double

visibility: public

class InvalidCouponException:

﻿ method definitions:

﻿ InvalidCouponException(String msg)

visibility: public

**Task**

Class **Product**

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

**-**define the **double** variable **price**

**-**define the **String** variable **coupon**

-define a parameterized constructor for all the data members.

Class **Validator**

Implement the below methods for this class:

-**String** **validateCoupon(Product p):**

* **throw** an **InvalidCouponException** "Invalid Coupon" if the coupon is not valid. **The coupon** is valid if its name and discount value are separated with '-' and the discount value should be between 10-25(inclusive).

**Example:**

**name**= "**IPhone**" ; valid **coupons** are "**IPhone-10**", "**IPhone-20**", "**IPhone-18**" etc.

* **return** "Valid Coupon" if no exception found.

-**double netPrice(Product p):**

* **netPrice** = totalPrice-discountPrice.
* **return** netPrice if Coupon is valid else **return** totalPrice.

Class **InvalidCouponException**

* define custom exception class **InvalidCouponException** by **extending** the **Exception** class.
* define a parameterised constructor with a String argument to pass the message to the super class.

**Sample Input**

**Product** obj = new Product("IPhone",25000,"IPhone-10");

**Validator** val = new Validator();

**String** valCop = val.validCoupon(obj);

**double** price = val.netPrice(obj);

**Sample Output**

valCop = "Valid Coupon"

price = 22500.0﻿

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q9

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 are public unless mentioned otherwise.

**Specifications**

class definitions:

﻿ class Customer:

﻿ Data members:

﻿﻿ name: String

﻿ mobileNo: String

﻿ cId: String

Define a parameterized constructor for all the data members.

class Validator:

﻿ validateCId(Customer c) throws Exception

﻿return type: String

visibility: public

validateMobileNo(Customer c) throws Exception

﻿ return type: String

visibility: public

class InvalidCIdException:

﻿ method definitions:

﻿ InvalidCIdException(String str)

visibility: public

class InvalidMobileNoException:

﻿ method definitions:

﻿ InvalidMobileNoException(String str)

visibility: public

**Task**

Class **Customer**

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

**-**define the **String** variable **mobileNo**

**-**define the **String** variable **cId**

-define a parameterized constructor for all the data members.

Class **Validator**

Implement the below methods for this class:

-**String** **validateCId(Customer c):**

* **throw** an **InvalidCIdException** "Invalid CId" if CId is not valid. **cId** is valid if it starts with first 4 character of **mobileNo** and end with last 2 character of **name**.

**Example:**

**name**= "Steve", **mobileNo** = "9898111111" then valid **cId** = "9898ve".

* **return** "Valid CId" if no exception found.

-**String validateMobileNo(Customer c):**

* **throw** an **InvalidMobileNoException** "Invalid MobileNo" if **mobileNo** is not valid. **mobileNo** is valid if it has 10 digits and starts with [9,8,7,6] only.
* **return** "Valid MobileNo" if no exception found.

Class **InvalidCIdException**

* define a custom exception class **InvalidCIdException** by **extending** the **Exception** class.
* define a parameterized constructor with a String argument to pass the message to the super class.

Class **InvalidMobileNoException**

* define a custom exception class **InvalidMobileNoException** by **extending** the **Exception** class.
* define a parameterized constructor with a String argument to pass the message to the super class.

**Sample Input**

﻿Customer obj = **new** Customer("Steve","9898111111","9898ve");

Validator val = **new** Validator();

String CID = val.validCId(obj);

String mob = val.validMobileNo(obj);

**Sample Output**

CID = "Valid CId"

mob = "Valid MobileNo"

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q10

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 are public unless mentioned otherwise.

**Specifications**

**class** **definitions**:

﻿ **class** **Student**:

Data members:

﻿﻿ name: String

﻿ score: **int**[]

status: String

Define a parameterized constructor **for** name **and** score data members.

**class** **Validator**:

﻿ validateScore(Student s) with throws Exception

﻿**return** type: String

visibility: **public**

getPercentage(Student s)

**return** **double**

visibility: **public**

class InvalidScoreException:

method definitions:

InvalidScoreException(String str)

visibility: **public**

**Task**

Class **Student**

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

**-**define the **int[]** variable **score**

**-**define the **String** variable **status**

-define a parameterized constructor for **the name and score** data member.

Class **Validator**

Implement the below methods for this class:

-**String** **validateScore(Student s):**

* **throw** an **InvalidScoreException** "Invalid score" if the score is not valid. Score is valid if **0<=score<=100**.
* **return** "Valid score" if no exception found.

-**double getPercentage(Student s):**

* **Percentage** = Sum of all the score divided by the total length of score.
* set the s.status to "pass" **if** percentage >= 33, **else** set s.status to "fail".
* **return** the percentage of the score if valid, else return 0.0.

Class **InvalidScoreException**

* define custom exception class **InvalidScoreException** by **extending** the **Exception** class.
* define a parameterized constructor with a String argument to pass the message to the super class.

**Sample Input**

﻿**int** score[] = {50,40,80,67,69};

Student s = **new** Student("Aman",score);

Validator val = **new** Validator();

String ans = val.validateScore(s);

**double** per =  val.getPercentage(s);

**Sample Output**

ans = Valid score

per = 61.2

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q11

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 are public unless mentioned otherwise.

**Specifications**

class definitions:

﻿ class SalaryData:

﻿Data members:

﻿﻿ name: String

﻿ daysInMonth: int

salary: double

﻿ Define a parameterized constructor for all Data members

class Validator:

﻿ validSalaryData(SalaryData s) throws Exception

﻿return type: String

visibility: public

totalSalary(SalaryData s)

﻿ return type: double

visibility: public

class InvalidDaysException:

﻿ method definitions:

﻿ InvalidDaysException(String str)

visibility: public

class InvalidSalaryException:

﻿ method definitions:

﻿InvalidSalaryException(String str)

visibility: public

**Task**

Class **SalaryData**

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

**-**define the **int** variable **dayInMonth**

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

-define a parameterized constructor for all the data members.

Class **Validator**

Implement the below methods for this class:

-**static** **String** **validSalaryData(SalaryData s):**

* **throw** an **InvalidDaysException** "Invalid Days" if daysInMonth is not valid. Here are the valid daysInMonth = [28,30,31].
* **throw** an **InvalidSalaryException** "Invalid Salary" if salary in invalid, salary is considered to be invalid if its less then 0 and greater than 10,00,000.
* **return** "Valid Data" if no above exception found.

-**static** **double totalSalary(SalaryData s):**

* **return** **daysInMonth\*salary** if SalaryData is valid else return 0.0.

Class **InvalidDaysException**

* define custom exception class **InvalidDaysException** by **extending** the **Exception** class.
* define a parameterized constructor with a String argument to pass the message to the super class.

Class **InvalidSalaryException**

* define custom exception class **InvalidSalaryException** by **extending** the **Exception** class.
* define a parameterized constructor with a String argument to pass the message to the super class.

**Sample Input**

﻿**SalaryData** s = new SalaryData("Steve",30,10000);

**Validator** val = new Validator();

**String** ans = val.validSalaryData(s);

**double** sal =  val.totalSalary(s);

**Sample Output**

ans = "Valid Data"

sal = 300000.0

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q12

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 are public unless mentioned otherwise.

**Specifications**

**class** **definitions**:

**class** **Student**:

data members:

StudentID: **int**

StudentName: string

visibility: **public**

**class** **InvalidStudentNameException**

**method** **definitions**:

﻿ InvalidStudentNameException(String s)

visibility: **public**

**class** **Source**

**method** **definitions**:

ValidateStudent(Student std):

**return** type: String

visibility: **public**

ValidateUserId(Student std):

**return** type: String

visibility: **public**

**Tasks**

Class Student

**-**define the **String** variable StudentName

**-**define the **String** variable StudentID

Class InvalidStudentNameException

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

Class Source

Implement the below methods for this class:

1. **String ValidateStudent(Student std):**

* This method validates the String StudentName argument and accordingly sets the value of StudentName variable available in Student std as below:
* Valid student name should start with a capital letter and rest should be small letters, **throw** a user-defined exception InvalidStudentNameException("Invalid Name") if StudentName is not valid.
* If the variable StudentName is a valid name ,**assign** the string "Valid" to **std.StudentName**
* **catch**the thrown exception and **assign** the string "Invalid" to **std.StudentName**
* **return** default message if an exception is thrown, else return **std.StudentName**

2. **String ValidateUserId(Student std):**

* This method validates the String StudentID argument and accordingly sets the value of StudentID variable available in Student std as below:
* A username is considered valid

-> If the username consists of 5 to 2**0** characters inclusive.

-> The username can only contain alphanumeric characters and underscores (\_).

-> The first character of the username must be an alphabetic character, i.e., either lowercase character **[a – z]** or uppercase character **[A – Z]**.

->should start with a capital letter and rest should be small letters

* **throw** a user-defined exception InvalidStudentNameException("Invalid Name") if StudentID is not valid.
* If the variable StudentIDis a valid StudentID ,**assign** the string "Valid" to **std.StudentID**
* **catch**the thrown exception and **assign** the string "Invalid" to **std.StudentID**
* **return** default message if an exception is thrown, else return **std.StudentID**

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q13

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.

**Input Format**

* The Main() method has already been implemented, which will pass values for the variables: **String1**, **String2** and **Operator**.

**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.

**Tasks:**

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

* Using try-catch block check that the value of either **string1**or **string2**variable is **null**, then throw appropriate exception for **NullPointerException** and return "**Null values found**".
* Using try-catch block check if the value of **operator**variable is not equal to "+" or "-" then throw and return the default exception(**Exception**) with the operator as parameter the return operator. Example if operator = "/" throw the **Exception(a.operator)**and return the operator: "/"**.**
* If no exception is found return "**No Exception Found**".

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

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

**Specifications:**

**class** **Activity**:

    data fields:

      String string1

      String string2

      String **operator**

    Implement a parameterized constructor to initialize all the instance variables

**class** **Source**:

method definitions:

handleException(Activity a): implement **try**-**catch** blocks **and** **throw** different exceptions as described in above.

**return** type: String

visibility: **public**

operate(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.
* **operate (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"**

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.

**ALLOWED TECHNOLOGIES**

* Java 8

Q14

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.

**Input Format**

* The **main()** method has already been implemented, which will pass values for the variables: **num1, num2 and op**.

**Tasks**

* Check that the value of **op** variable is equal to '\*' and the value of either **num1** or **num2** is 0, then throw appropriate exception for **MultiplyByZeroException** (described in **Output Format**).
* Check that the value of **op** variable is equal to ‘**/**’ and the value of **num2** variable is 0, then throw appropriate exception for **DivideByZeroException**(described in **Output Format**).
* Check if the value of **op** variable is not equal to arithmetic operator ((+, -, \* or /) using logical AND operator. If the condition is true then throw appropriate exception (described in **Output Format**).
* Using **result** variable, perform the **calculate()** function by passing the value **num1**, **num2** and **op** variables as an argument.
* In **calculate()** function perform the arithmetic operations, using **switch** statement.

**Output Format**

* Depending on the exception raised, return the following:
* For **MultiplyByZeroException**, return the message "Multiplication with zero results in zero"
* For **DivideByZeroException**, return the message "Division by zero results in infinity"
* For any other exceptions, return the message "\_\_\_ is not a valid operator", where \_\_\_ represents the variable **op**
* If no exceptions are raised, **return** the message "No Exception Found" and the **result** of the arithmetic operation as a **double** value.
* If an exception is raised, make sure the **result** of the arithmetic operation should be returned as **0.0**. Hence, handle the arithmetic operations accordingly.
* Validate the above in the **calculate()** method also.

**Specifications**

**class** **definitions**:

Exceptions:

method definitions:

checkException(**double** n1, **double** n2, **char** op): implement **try**-**catch** blocks **and** **throw** different exceptions as described in Output Format

**return** type: String

visibility: **public**

calculate(**double** v1, **double** v2, **char** op): implement **switch** statement to calculate result based on value of op

**return** type: **double**

visibility: **public**

DivideByZeroException: will extend Exception class

method definitions:

DivideByZeroException(String s): call constructor of parent Exception

MultiplyByZeroException: will extend Exception class

method definitions:

MultiplyByZeroException(String s): call constructor of parent Exception

**NOTE:**

* The methods **checkException()** and **calculate()** will be called as part of the test-cases in all scenarios.
* Take care in defining the return values. For instance, the **result**value of **0.0** should be returned for the exception **DivideByZeroException** or **MultiplyByZeroException**.

**Sample Test Cases**

**Sample Input #1**

10

,

12

**Sample Output #1**

, **is** **not** a valid operator

**Sample Input #2**

10

+

12

**Sample Output #2**

No Exception Found

22.0

**Sample Input #3**

1

/

0

**Sample Output #3**

Division **by** zero results **in** infinity

**ALLOWED TECHNOLOGIES**

* Java 8

## Q15

## **Environment Specifications & Instructions**

Allowed Languages

* Java7
* Java8

**Build Expectation**

Problem Statement

Marcus is an avid programmer by choice and a transporter by profession. To create and automate a database that would list his current vehicles, he decides to program a system for it. You, being his work partner, have been allotted the job of developing the back-end of this system. The exact task description is as follows:

Task

Create an exception named ***TypeException.***

Overrides the toString() method:

It will **return** a String "Vehicle Type Not Set".

You need to create a class named ***Vehicles***:

Data members (All Strings):

type, model\_no, model\_name, owner\_name, owner\_details

Constructor that will take following parameters as input and will **set** them:

model\_no, model\_name, owner\_name, owner\_details

**And** a **default** **constructor** too.

**Following** methods:

get\_type: **returns** the **value** **of** **type**

retrieve: **returns** a string. Here, will **return** "null" **as** String.

Another class named ***Car*** which will inherit ***Vehicles*** class:

Constructor that will take following parameters as input and will **set** them:

Will **call** super **Constructor** **with** the **input** parameters.

**And** a **default** **constructor** too.

**Following** methods:

get\_type: **returns** the **value** **of** **type**

﻿ set\_type: setter method **with** void **return** **type**

retrieve:

Overrides the Vehicles retrieve method.

**if** **type** **is** null, it throws a TypeException.

**else** it **returns** the **string** concatenating model\_no, model\_name, owner\_details **and** owner\_name

**NOTE:**

* The **main() method** has been implemented as part of the test-cases. Hence, it does NOT NEED to be implemented.
* You can implement the **main()** method, inside a class named **Source** and call all methods there, for self-evaluation. Your implementation can be then tested out using the **RUN CODE** button.
* In case, you haven't implemented the **main() method**, you can only click on the buttons **VERIFY** and **SUBMIT**.
* In the absence of a **main() method** in your implementation, clicking on **RUN CODE** will result in an error.

**ALLOWED TECHNOLOGIES**

* Java 7
* Java 8

Q16

Complete the class**Student**using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

**class** **definition**:

﻿**class** **Student**:

data field:

﻿HashMap<String,String> dataList

method definitions:

addStudentDetails(String name, String id):

﻿**return** type: String

deleteId(String name):

**return** type: String

searchId(String name):

**return** type: String﻿

**Task**

Class **Student**

**-**define the **HashMap<String, String>**variable **dataList.**

* Keys- String -> **name**
* Values- String -> **id**

Implement the below methods for this class:

**-String addStudentDetails(String name, String id):**

* Write a code to add students detail to the **map**.
* If the name is already present in the **map**then return**"Id generation Failed, name is already present in the database".**
* If the name is not there in the **map**then add data in **map**and return **"Id Generated".**

**-String deleteId(String name):**

* Write a code to delete the student detail from the **map**.
* If the student name is not found in the map then return **"Id not found".**
* If the student name is present in the **map**then delete the student details and return **"Id deleted successfully".**

**-String searchId(String name):**

* Write a code to search the student id from the **map**.
* If the student's name isn't found in the **map**then return**"Id not found".**
* If the student's name is present in the **map**then delete the student details and return **id**.

**Sample Input**

s1 = obj.addStudentDetails("Sarah","99889");

s2 = obj.addStudentDetails("John","78985");

s3 = obj.deleteId("Sarah");

s4 = obj.searchId("Steve");

**Sample Output**

"Id Generated"

"Id Generated"

"Id deleted successfully"

"Id not found"

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

**TAGS**

* Hashmap

Q17

**DESCRIPTION**

**Problem Statement**

You are given an integer, return its base 7 string representation.

**Input Format**

* Single line containing an integer, **n**

**Constraints**

* -1000 <= **n** <= 10000

**Output Format**

* Single line returning the base 7 string representation

**Evaluation Parameters**

* **Sample Input**

201

* **Sample Output**

405

**ALLOWED TECHNOLOGIES**

*All available technologies for coding problems enabled.*

**TAGS**

* Maths
* Base 7
* String representation

Q18

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Material:

int ﻿mass

int temperature

visibility : public

Material(int mass, int temperature) : Constructor **with** **public** visibility

﻿method definitions:

﻿flowOfHeat() throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿ checkMaterial() throws **Exception**:

**return** **type** : **String**

visibility : **public**

﻿

**class** MaterialException extends **Exception**:

﻿method definitions:

﻿ MaterialException(**String** msg)

visibility: **public** ﻿

**Task**

**Class** **Material**

**-define all the data members as per the given specifications.**

**-define the constructor with public visibility.**

**-Implement the below methods for this class:**

**-String** **checkMaterial() throws Exception:**

* Write a code to check whether the material is valid or not.
* **throw a MaterialException** if the mass is negative with a message "**Mass cannot be negative**".
* **throw a MaterialException** if the mass is zero with a message "**Mass cannot be zero**"
* if no exception is found return "**Valid material"**

**-String** **flowOfHeat() throws Exception:**

* If **checkMaterial()** method throws a **MaterialException** then returns a message "**Invalid Material**".(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* **If no exception is found then return "Both in equilibrium".**

**Sample Input**

Material m=**new** Material(12,32);

String s=m.flowOfHeat();

**Sample Output**

**Both in equilibrium**

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

**TAGS**

* Exceptions
* Exceptions and Exception Handling

Q19

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Cue:

data members:

int pieces

boolean retain

Cue(int pieces, boolean retain): constructor **with** **public** visibility

**class** **Retention**:

Cue(Cue c): **constructor** **with** **public** visibility

**data** members:

Cue cue=null

method definitions:

checkCue(int p) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

playGame(int p) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** CueException extends **Exception**:

method definitions:

CueException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Cue**

**-**define the **int**variable **pieces.**

**-**define the **boolean** variable **retain**

**-**define a **constructor** according to the above specifications.

Class **Retention**

**Define the class according to the above specififcations and Implement the below methods for this class:**

**-String** **checkCue(int p) throws Exception:**

* Write a code to validate the criteria for getting the award.
* **throw a CueException**if **retain** is falsewith the message "**Cue not retained**".
* **throw a CueException**if **p is** less than pieces of cue variable with the message "**More pieces required**".
* throw a CueException if p is greater than pieces of cue variable with the message **"Update required".**
* If no above exception is found then return a string message "**Cue updated**".

**-String playGame(int p) throws Exception:**

* Write a code to play the game using the mentioned cue.
* If **checkCue()** method throws a **CueException** then returns a message "**Cannot use this cue**".(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* If no exception is found then return a message "**Welcome to the game**".

**class CueException extends Exception**

* Define **CueException** class derived from Exception class

**Sample Input**

Cue c=**new** Cue(13,true);

Retention r= **new** Retention(c);

String ans = r.playGame(5);

**Sample Output**

cannot use **this** cue

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

**TAGS**

* Exceptions
* Errors & Exceptions

Q20

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 Student

   data member:

     String stu\_name;

     int score;

  Student(String stu\_name, int score): constructor **with** **public** visibility

**class** Merit:

**data** **member**:

﻿﻿ HashMap<**String**, ArrayList<Student>> mlist= **new** HashMap<>()

method definitions:

newEntry(Student s, **String** university)

**return** **type**: **String**

visibility: **public**

getStudents(**String** university)

**return** **type**: ArrayList<**String**>

visibility: **public**

**Task**

class **Student**

-Define this class according to the above specifications

**Class** **Merit**

**-define the object of HashMap<University, ArrayList<Student>>** with variable name **mlist**.

**-String newEntry(Student s, String university):**

* Write a code to check if the university is present in the hashmap or not.
* If present, then append the given student object to the list with key as the university and return "**Student added**".
* If not present then add this university as a key and a new list with student object added into it and return "**University added**"

**-ArrayList<String> getStudents(String university):**

* Write a code to get the list of the students' names who are enrolled in the university passed in the argument.
* If there are no students then return null.

**Sample Input**

Merit obj = **new** Merit();

Student s1=**new** Student("s1",100);

obj.newEntry(s1,"IIT BOMBAY");

**Sample Output**

**University** added

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q21

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Car:

﻿data members:

int carId

int speed

visibility : public

Car(int carId, int speed): constructor **with** **public** visibility

**class** CarRacing:

﻿**data** members:

**String** ArrayList<Car> cars

int trackCount

visibility : **public**

﻿

CarRacing(int trackCount): **constructor** **with** **public** visibility

method definitions:

carEntry(Car car):

**return** **type**: **String**

visibility: **public**

﻿

getWinner(int trackLength):

**return** **type**: int

visibility: **public**

**Task**

**Class Car**

**-** define all the variables according to the above specifications.

- define a**constructor** with public visibility.

**Class CarRacing**

**-** define all the variables according to the above specifications.

**-** define

* participants as **ArrayList<Car> cars = new ArrayList<>();**

- define a**constructor** with public visibility.

**Implement the below methods for this class:**

**-String** **carEntry(Car car):**

* Write a code that adds the given **car**to the list of **cars**according to the given conditions.
* If the trackCount is equal to the length of the list **cars**then return **"No Space".**
* If the list **cars**have already an entry with the **same id** as the given parameter **car**then return **"Already exists".**
* If both the above condition does not satisfy then add the given parameter **car**into the list of **cars**and return **"Start practicing".**

**-int** **getWinner(int trackLength):**

* Write a code that returns the**carId of the winner.**
* If no entry exists then return **-1.**
* Else return the carId of the **car** with maximum **time.**
* The formula to calculate the **time = trackLength \* speed.**
* If two cars have the same time then return the carId which is added to the list of **cars**first.

**Sample Input**

Car car = **new** Car(1,100);

CarRacing carRacing = **new** CarRacing(2);

carRacing.carEntry(car);

**Sample Output**

**Start** practicing

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q22

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** **Dance**:

data member:

﻿﻿ HashMap<String, Integer> steps= **new** Hashmap<>()

﻿ method definition:

newStep(String name, **int** ways)

**return** type: String

visibility: **public**

findStep(String name)

**return** type: **int**

visibility: **public**

**Task**

**Class** **Dance**

**-define the object of HashMap<String, Integer>** with variable name **steps**.

* The **String**defines the **name of the step**and the **integer**will have number of ways it is performed.

**Implement the below methods for this class:**

**-String newStep(String name, int ways):**

* Write a code to add **a new step** in the **number of ways**.
* If the step does not exist then create it and add the step to the hashmap and return **"New step"**.
* Update the number of ways if the name already exists
* return "**Stepped in**" after performing the above operations if it already exists.

**-int findStep(String name):**

* Write a code to return the number of ways a particular step can be performed.
* If it is not present then return 0 else return the number of ways.

**Sample Input**

Dance obj = **new** Dance();

obj.newStep("Step1",9);

**Sample Output**

**New** step

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q23

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** Racer:

data member:

String action

**int** runScore

**int** jumpScore

**int** crawlScore

visibility: public

﻿

﻿﻿ Racer(String action, **int** runScore, **int** jumpScore, **int** crawlScore):constructor with public visibility

method definition:

﻿ goodAt():

**return** : String

﻿ visibility : public

﻿finalScore():

**return** : **int**

﻿ visibility : public﻿

Task

**class Racer**

- define **data members** according to the above specifications

-define a constructor according to the above specifications

-The term/variable used are defined below -

1. **action** - A string that contains only 3 different characters **'r', 'j', and 'c'**. Where **'r' - run, 'c' - crawl, and 'j' - jump.**
2. **runScore** - A int that denotes what score will he get for every 'r' in **action** string.
3. **jumpScore** - A int that denotes what score will he get for every 'j' in **action** string.
4. **crawlScore** - A int that denotes what score will he get for every 'c' in **action** string.

**-Implement the below methods for this class:**

**-String goodAt():**﻿

* Write a code that returns the string on the basis of the given conditions -
* If a count of the character 'j' is equal to the count of the character 'c' in the string **action**then return **"Perfect".**
* If a count of the character 'j' is greater than the count of the character 'c' in the string **action**then return **"Jumper".**
* If a count of the character 'j' is less than the count of the character 'c' in the string **action**then return **"Crawler".**

**-int finalScore():**

* Return the sum of the score for each action.

1. For every **'r'** in action, the score will be **runScore.**
2. For every**'j'** in action, the score will be**jumpScore.**
3. For every**'c'**in action, the score will be **crawlScore.**

* Refer Example for better understanding

***action = "jjcr" , runScore = 20, jumpScore = 30, crawlScore = 5***

***action.goodAt()****returns****"Jumper"****as****'j' count > 'c' count****in****action.***

***action.finalScore()****returns****85****as****20+20+5+30 = 85.***

**Sample Input**

Racer racer = **new** Racer("jjccrrj",10,20,30);

racer.goodAt();

**Sample Output**

**Jumper**

**NOTE:**

* You can make suitable function calls and use **RUN CODE** button to check your **main()** method output.
* Make sure that all the strings in the return statement are case sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q24

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:**

enum definition:

enum Specialty

ENGINEER, MANAGER, DEVOPS

class definitions:

class Specialist:

data members:

String name;

BigDecimal salary;

Specialty specialty;

visibility: private

Specialist(String name, BigDecimal salary, Specialty specialty): constructor **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString(): has been implemented **for** you

**class** Implementation:

method definition:

filterBySpecialty(**List**<Specialist> specialists, Specialty specialty):

**return** **type**: **List**<Specialist>

visibility: **public**

sortSpecialistsByNameAsc(**List**<Specialist> specialists):

**return** **type**: **List**<Specialist>

visibility: **public**

**Task:**

**enum Specialty:**

**-Define enum** according to the above specifications

**class Specialist:**

**-**Define the class according to the above specifications

class **Implementation:**

**Implement the below method for this class**:

* **Specialist findWithMaxSalary(List<Specialist> specialists):**return the details of the specialist with the maximum salary
* **boolean matchAllEngineers(List<Specialist> specialists):**return **true** if all the specialist have **speciality = ENGINEER**, else return **false**

*Refer sample output for clarity*

**Sample Input**

**List**<Specialist> **list** = Arrays.asList(

**new** Specialist("Ivan Ivanov", **new** BigDecimal(5000), Specialty.ENGINEER),

**new** Specialist("Sergey Sergeev", **new** BigDecimal(3000), Specialty.DEVOPS),

**new** Specialist("Petr Petrov", **new** BigDecimal(10000), Specialty.MANAGER)

        );

**Sample Output**

Specialist{name='Petr Petrov', salary=10000, specialty=MANAGER}

-----------------------------------------------------------

false

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q25

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:**

enum definition:

enum Specialty

ENGINEER, MANAGER, DEVOPS

class definitions:

class Specialist:

data members:

String name;

BigDecimal salary;

Specialty specialty;

visibility: private

Specialist(String name, BigDecimal salary, Specialty specialty): constructor **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString(): has been implemented **for** you

**class** Implementation:

method definition:

filterBySpecialty(**List**<Specialist> specialists, Specialty specialty):

**return** **type**: **List**<Specialist>

visibility: **public**

sortSpecialistsByNameAsc(**List**<Specialist> specialists):

**return** **type**: **List**<Specialist>

visibility: **public**

**Task:**

**enum Specialty:**

-Define **enum** according to the above specifications

**class Specialist:**

**-**Define the class according to the above specifications

class **Implementation:**

**Implement the below method for this class**:

* **List<Specialist> filterBySpecialty(List<Specialist> specialists, Specialty specialty):** filter the list by Speciality and return the list
* **List<Specialist> sortSpecialistsByNameAsc(List<Specialist> specialists):**sort and return the list by name

*Refer sample output for clarity*

**Sample Input**

**List**<Specialist> **list** = Arrays.asList(

**new** Specialist("Ivan Ivanov", **new** BigDecimal(5000), Specialty.ENGINEER),

**new** Specialist("Sergey Sergeev", **new** BigDecimal(3000), Specialty.DEVOPS),

**new** Specialist("Petr Petrov", **new** BigDecimal(10000), Specialty.MANAGER));

**Sample Output**

[Specialist{name='Ivan Ivanov', salary=5000, specialty=ENGINEER}]

-----------------------------------------------------------

[Specialist{name='Ivan Ivanov', salary=5000, specialty=ENGINEER}, Specialist{name='Petr Petrov', salary=10000, specialty=MANAGER}, Specialist{name='Sergey Sergeev', salary=3000, specialty=DEVOPS}]Customer{name='Tom', age=30}]

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

**TAGS**

* StreamAPI
* Lambda

Q26

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** **Member**:

method definition:

match(List<String> memberNames):

**return** type: List<String>

visibility: **public**

countA(List<String> memberNames):

**return** type: **long**

visibility: **public**

**Task:**

class **Member:**

**Implement the below method for this class**:

* **List<String> match(List<String> memberNames):**  sort the list alphabetically , toggle the Strings to uppercase and return the list
* **long countA(List<String> memberNames):** count and return the number of **A** present in the list

*Refer sample output for clarity*

**Sample Input**

List<String> memberNames = **new** ArrayList<>(Arrays.asList("Amitabh", "Shekhar", "Aman", "Rahul",

"Shahrukh", "Salman", "Yana", "Lokesh"));

**Sample Output**

2

---------------------------------------------------------

[AMAN, AMITABH, LOKESH, RAHUL, SALMAN, SHAHRUKH, SHEKHAR, YANA]

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q27

our 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 Customer:

data members:

String name;

    int age;

visibility: private

Customer(String name, int age): constructor **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString(): has been implemented **for** you

**class** Implementation:

method definition:

sortCustomerList(**List**<Customer> customersList):

**return** **type**: **List**<Customer>

visibility: **public**

withDelimeters(**List**<Customer> customersList):

**return** **type**: **String**

visibility: **public**

**Task:**

**class Customer:**

**-**Define the class according to the above specifications

class **Implementation:**

**Implement the below method for this class**:

* **String** **withDelimeters(List<Customer> customersList)**:  concat and return the customer details with delimeter **"@"**
* **List<Customer> sortCustomerList(List<Customer> customersList):**sort the list lexicographically and return it

*Refer sample output for clarity*

**Sample Input**

**List**<Customer> customersList = Arrays.asList(

**new** Customer("Tom", 30),

**new** Customer("Steve", 31),

**new** Customer("Peter", 34),

**new** Customer("Simon", 23));

**Sample Output**

Customer{name='Tom', age=30}@Customer{name='Steve', age=31}@Customer{name='Peter', age=34}@Customer{name='Simon', age=23}

---------------------------------------------------------

[Customer{name='Peter', age=34}, Customer{name='Simon', age=23}, Customer{name='Steve', age=31}, Customer{name='Tom', age=30}]

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q28

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 Product:

data members:

Integer id;

    String name;

    Double price;

visibility: private

Product(Integer id, String name, Double price): constructor **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString(): has been implemented **for** you

**class** Implementation:

method definition:

getProductName(**List**<Product> products):

**return** **type**: **Map**<**String**, **List**<Product>>

visibility: **public**

getSum(**List**<Product> products):

**return** **type**: **Map**<**String**, **Double**>

visibility: **public**

**Task:**

**class Product:**

**-**Define the class according to the above specifications

class **Implementation:**

**Implement the below method for this class**:

* **Map<String, List<Product>> getProductName(List<Product> products):**Filter and return the product details with price greater than **20**
* **Map<String, Double> getSum(List<Product> products):**  return the **price** of the products

*Refer sample output for clarity*

**Sample Input**

Product pr5 = **new** Product(1, "Bruchotte", 5.0);

Product pr6 = **new** Product(4, "Banana", 1305.0);

**List**<Product> pr= Arrays.asList(pr5, pr6);

**Sample Output**

{Banana=[Product{id=4, name='Banana', price=1305.0}]}

---------------------------------------------------------

{Bruchotte=5.0, Banana=1305.0}

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q29

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 Program:

﻿data members:

String name

boolean value

ArrayList<String> tags

visibility: public

﻿﻿ Program(String name, boolean value, ArrayList<String> tags): constructor with public visibility

﻿

class Menu:

﻿ data members :

Program prog

visibility : public

Menu(Program p): constructor with public visibility﻿

﻿method definition:

addProgram():

return : String

visibility public

﻿runQuery(String query):

﻿ return : String

﻿ visibility : public

**class Program**

- define data members according to the above specifications

-define a constructor and getters setters according to the above specifications

**class Menu**

- define data members according to the above specifications

-Implement the below methods for this class:

**-String addProgram():**

* ﻿Write a code to check if the boolean value is true.
* If it is false then set the value to true and return "**Boolean changed".**
* **Convert the name into title form i.e. every word much have the first character in uppercase and return "Program ready".**

**-String runQuery(String query):**

* The query contains some comma-separated strings(,).
* Check if those strings are present in the tags ArrayList of a prog variable.
* If they are present then return "**No changes"**.
* Else return the number of tags added from the query string along with the given message format. (For example, there are two tags added then return "**2 tag(s) added".**

**Sample Input**

Program p = **new** Program("reconcile program", true, **new** ArrayList<String>(Arrays.asList("C","D")));

Menu m = **new** Menu(p);

m.addProgram();

m.runQuery("A,B");

**Sample Output**

Program ready

2 tags(s) added

**NOTE:**

* You can make suitable function calls and use the **RUN CODE** button to check your **main()** method output.
* Make sure that all the strings in the return statement are case-sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q30

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Stocks:

float stockRate

visibility : public

Stocks(float stockRate) : Constructor **with** **public** visibility

﻿method definition:

﻿checkRate(float Rate) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** StockHighException extends **Exception**:

﻿method definition:

﻿ StockHighException(**String** msg)

visibility: **public**

﻿

**class** StockLowException extends **Exception**:

﻿method definition:

﻿ StockLowException(**String** msg)

visibility: **public** ﻿

**Task**

**Class** **Stocks**

-define all the data members as per the given specifications.

-define the constructor with public visibility.

**-Implement the below methods for this class:**

**-String** **checkRate(float rate) throws Exception:**

* Write a code that checks for the Profit and Loss in the stock rate**.**
* If the given parameter rate is 80% of the stockRate then return a string "Hope to raise".
* If the given parameter rate is less than 80% of the stockRate then throw StockLowException with a message "Under Loss".
* If the given parameter rate is greater than 80% of the stockRate then throw StockHighException with a message "Under Profit".

**Class** **StockHighException extends Exception:**

-Define**StockHighException** class derived from the Exception class.

**Class** **StockLowException extends Exception:**

-Define**StockLowException** class derived from the Exception class.

**Sample Input**

Stocks stock = **new** Stocks(10.0f);

stock.checkRate(5.0f);

**Sample Output**

**StockLowException** : Under Loss

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q31

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 Dog:

data members:

String name

    int age

    int weight

Dog(String name, int age, int weight): constructor **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString(): has been implemented **for** you

**class** Implementation:

method definition:

filterByAgeAndWeight(**List**<Dog> listDog):

**return** **type**: **List**<Dog>

visibility: **public**

separateWithDelimeter(**List**<Dog> listDog):

**return** **type**: **String**

visibility: **public**

**Task:**

**class Dog:**

**-**Define the class according to the above specifications

class **Implementation:**

**Implement the below method for this class**:

* **List<Dog> filterByAgeAndWeight(List<Dog> listDog):**fetch dog details on the basis of:
* **age** greater than **10**
* **weight** greater than **25**
* get the filtered data, put it into a list and return the list
* **String separateWithDelimeter(List<Dog> listDog):** concat and return the dogs details with delimeter **"**$~$~**"**

*Refer sample output for clarity*

**Sample Input**

**List**<Dog> **list** = **new** ArrayList<Dog>();

**list**.add(**new** Dog("German Shepherd ", 20, 35));

**list**.add(**new** Dog("Labrador ", 5, 40));

**list**.add(**new** Dog("Pitbull ", 29, 100));

**list**.add(**new** Dog("Poodle", 10, 45));

**Sample Output**

[Dog{name='German Shepherd ', age=20, weight=35}, Dog{name='Pitbull ', age=29, weight=100}]

----------------------------------------------------------

Dog{name='German Shepherd ', age=20, weight=35}$~$~Dog{name='Labrador ', age=5, weight=40}$~$~Dog{name='Pitbull ', age=29, weight=100}$~$~Dog{name='Poodle', age=10, weight=45}

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q32

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 Product

data members:

  Integer id;

    String name;

    Double price;

visibility: private

 Product(Integer id, String name, Double price): constructor **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString() method has been implemented **for** you

**class** Implementation:

method definition:

getProductName(**List**<Product> products) :

**return** **type**: **Map**<**String**, **List**<Product>>

visibility: **public**

getSum(**List**<Product> products) :

**return** **type**: **Map**<**String**, **Double**>

visibility: **public**

**Task:**

**class Product:**

- define the class according to the above specifications

**class Implementation:**

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

* **Map<String, List<Product>> getProductName(List<Product> products):**

fetch and return the **details** of all the products

* **Map<String, Double> getSum(List<Product> products)**:  **sum** all the product in the list and return it

**Sample Input**

Product pr1 = **new** Product(1, "Ceviche", 15.0);

Product pr2 = **new** Product(2, "Chilaquiles", 25.50);

Product pr3 = **new** Product(3, "Bandeja Paisa", 35.50);

Product pr4 = **new** Product(4, "Ceviche", 15.0);

**List**<Product> pr = Arrays.asList(pr1, pr2, pr3, pr4);

Implementation imp = **new** Implementation();

------------------------------------------------------

imp.getProductName(pr)

imp.getSum(pr)

**Sample Output**

{Bandeja Paisa=[Product{id=3, name='Bandeja Paisa', price=35.5}], Chilaquiles=[Product{id=2, name='Chilaquiles', price=25.5}]}

---------------------------------------------------------

{Ceviche=30.0, Bandeja Paisa=35.5, Chilaquiles=25.5}

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q33

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 Person

data members:

  Integer id

    String name

    LocalDate birthDate

visibility: private

Person(Integer id, String name, LocalDate birthDate): constructor **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString() method has been implemented **for** you

**class** Implementation:

method definition:

filterListByBirth(**List**<Person> persons) :

**return** **type**: **List**<Person>

visibility: **public**

limitSkipAndReturn(**List**<Person> persons, int pageNumber, int pageSize) :

**return** **type**: **List**<Person>

visibility: **public**

**Task:**

**class Person:**

- define the class according to the above specifications

**class Implementation:**

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

* **List<Person> filterListByBirth(List<Person> persons):**filter and return the list by**date of birth**
* **List<Person> limitSkipAndReturn(List<Person> persons, int pageNumber, int pageSize):** get the multiplication of pageNumber and pageSize, skip those indexes, now limit the pageSize and return the list

**Example:** For the below list in the sample input, page number \* page size = 2, skip the first 2 indexes in the list, now limit the page size = 2, after limiting the page size we get the desired result as given below in sample output

*Refer to the sample input output for more clarifications*

**Sample Input**

 Person p1 = **new** Person(1, "Mito", LocalDate.**of**(1991, 1, 21));

 Person p2 = **new** Person(2, "Code", LocalDate.**of**(1990, 2, 21));

 Person p3 = **new** Person(3, "Jaime", LocalDate.**of**(1980, 6, 23));

 Person p4 = **new** Person(4, "Duke", LocalDate.**of**(2019, 5, 15));

 Person p5 = **new** Person(5, "James", LocalDate.**of**(2010, 1, 4));

List<Person> persons = Arrays.asList(p1, p2, p3, p4, p5);

------------------------------------------------------

imp.filterListByBirth(persons)

imp.limitSkipAndReturn(persons, 1, 2)

**Sample Output**

[Person{id=3, name='Jaime', birthDate=1980-06-23}, Person{id=2, name='Code', birthDate=1990-02-21}, Person{id=1, name='Mito', birthDate=1991-01-21}, Person{id=5, name='James', birthDate=2010-01-04}, Person{id=4, name='Duke', birthDate=2019-05-15}]

---------------------------------------------------------

[Person{id=3, name='Jaime', birthDate=1980-06-23}, Person{id=4, name='Duke', birthDate=2019-05-15}]

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q34

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Bomb:

data members:

int time

String color

Bomb(int time, String color): constructor **with** **public** visibility

**class** Suicide:

**data** members:

Bomb bomb

Suicide(Bomb bb): **constructor** **with** **public** visibility

method definitions:

diffuseIt(int time, **String** color) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

checkSafety(int time, **String** color) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** SuicideException extends **Exception**:

method definition:

SuicideException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Bomb**

**-**define the **int**variable **time.**

-define **String** variable **color**

**-**define a **constructor** according to the above specifications.

Class **Suicide**

-define the bomb variable with null;

**-**define a **constructor** according to the above specifications and initialize the bomb variable with the object passed in the argument.

**Implement the below methods for this class:**

**-String** **diffuseIt(int time, String color) throws Exception:**

* Write a code to validate the criteria for getting the award.
* **throw a SuicideException**if **time**is greater than **the time of the bomb**with the message "**Time exceeded**".
* **throw a SuicideException**if the **color of the bomb is different from the color passed as an argument** with the message "**Wrong color**".
* If no above exception is found then return a string message "**Hope is there**".

**-String checkSafety(int time, String color) throws Exception:**

* Write a code to send an invite to the nominee.
* If **diffuseIt()**method throws a **SuicideException** then returns a message "**Bomb exploded**".(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* If no exception is found then return a message "**Take a bow**".

**Sample Input**

Bomb b=**new** Bomb(10,"red");

Suicide sc=**new** Suicide(b);

String s = sc.diffuseIt(5,"red");

String t = sc.checkSafety(8,"blue");

s.toLowerCase();

t.toLowerCase();

**Sample Output**

hope is there

bomb exploded

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q35

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:

 class Task:

  data members:

    String name

    int hours

  Task(String name, int hours): constructor **with** **public** visibility

**class** ToDoList:

**data** members:

**List**<Task> tasks

 method definitions:

   addTask(Task t) throws **Exception**:

**return** **type**: **String**

    visibility: **public**

   completeTask(Task t) throws **Exception**:

**return** **type**: **String**

    visibility: **public**

**class** TaskException extends **Exception**:

  method definition:

   TaskException(**String** msg)

     visibility: **public**

**Task**

Class **Task**

- define the String variable name.

- define the int variable hours

-define a constructor according to the above specifications.

Class **ToDoList**

Implement the below methods for this class:

-**String** **addTask(Task t) throws Exception:**

* Write a code to validate the criteria for getting the award.
* throw a **TaskException**if '**hours**' is less than 1 or greater than 24 with the message "**Invalid time**".
* throw a **TaskException**if the **given object**is already present in the ArrayList with the message "**Already present**".
* If no above **exception**is found then add the given task to the given ArrayList and return a string message "**Task will be completed**".

-**String completeTask(Task t) throws Exception**:

Write a code to complete the task.

* If **addTask**() method throws a TaskException then returns a message "**Task incomplete"**.(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* If no exception is found then return a message "**Task completed**".

**Sample Input**

Task t= **new** Task("Gym",12);

ToDoList lst=**new** ToDoList();

String t1=lst.addTask(t);

t1.toLowerCase();

**Sample Output**

**task** will be completed

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q36

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 Student:

data members:

String name

int roll

﻿

﻿ Student(String name, int roll): constructor **with** **public** visibility

**Define** getter **and** setter **with** **public** visibility﻿

**class** Record:

**data** members :

**List**<Student> students

﻿

method definition:

addStudent(Student student):

**return** : **String**

visibility **public**﻿

swapRolls(Student s1, Student s2):

**return** : **String**

visibility : **public**

**class Student**

- define data members according to the above specifications

-define a constructor and getters setters according to the above specifications

**class Record**

- define data members according to the above specifications

-Implement the below methods for this class:

**-String addStudent(Student student):**

* ﻿Write a code to add a given room object to the students when the name is valid.
* The name is said to be valid if it contains only alphabetical characters.
* Return "**Added**" when the name is valid else return "**Invalid name".**

**-String swapRolls(Student s1, Student s2):**

* Given two students, swap the roll numbers of these and return "**Swapped**".

**Sample Input**

Student s1=**new** Student("A",1);

Record record = **new** Record();

record.addStudent(s1);

**Sample Output**

**Added**

**NOTE:**

* You can make suitable function calls and use the **RUN CODE** button to check your **main()** method output.
* Make sure that all the strings in the return statement are case-sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q37

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Player:

data members:

int requiredRuns

String summary

Player(int requiredRuns, String summary): constructor **with** **public** visibility

**class** **Match**:

method definitions:

checkTarget(Player player) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

results(Player player) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** MatchException:

method definitions:

MatchException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Player**

**-**define the **int**variable **requiredRuns.**It denotes the runs required in that particular over.

**-**define the **String** variable **summary**

**-summary**defines the way that particular over was bowled. It contains runs like **'1', '2', '3', '4', and '6'.**If the player gets out then it is denoted by**'W'.**

**-**define a **constructor** according to the above specifications.

Class **Match**

**Implement the below methods for this class:**

**-String** **checkTarget(Player player) throws Exception:**

* Write a code to validate the criteria for getting the award.
* **throw** a **MatchException** if the player gets out with the message "**Out**".
* **throw a MatchException**if **the number of runs in that over is less than required runs**with the message "**Lost by runs**".
* If no above exception is found then return a string message "**We may win**".

**-String results(Player player)throws Exception:**

* Write a code to send an invite to the nominee.
* If **checkTarget** method throws a **MatchException** then returns a message "**We lost**".(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* If no exception is found then return a message "**We won**".

**class MatchException extends Exception**

* Define **MatchException** class derived from Exception class

**Sample Input**

Player p=**new** Player(12,"123461");

Match m=ne Match();

String s = m.checkTarget(p);

String t = m.results(p);

s.toLowerCase();

t.toLowerCase();

**Sample Output**

we may win

we won

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q38

Write a java program to find the number of words in a given sentence.

**Evaluation Parameters**

* **Sample Input**

String s1="Hello India"

* **Sample Output﻿**

2

**ALLOWED TECHNOLOGIES**

* Java 7
* Java 8

Q39

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 Student:

data members:

String name

int score

﻿﻿ Student(String name, int score): constructor **with** **public** visibility

**class** Classroom:

﻿ method definition:

registerStudent(Student student):

**return** : **String**

visibility: **public**

studentCard(**String** card):

**return** : **String**

﻿visibility : **public**

**class Student**

- define data members according to the above specifications

**class Classroom**

- define data members according to the above specifications

-Implement the below methods for this class:

**-String registerStudent(Student student):**

* ﻿Write a code to validate the names and marks of the students according to the below specifications.
* The name must be in uppercase, if not return "**Block letters needed**".
* If the score is not between 0 to 100 then return **"Invalid score".**
* **If the above conditions are satisfied**then return "**Registered**".

**-String studentCard(String card):**

* The given string contains **only numbers**(no alphabets and special symbols allowed).
* If it has any other character return "**Invalid card" else return "Valid card".**

**Sample Input**

 Student s1=**new** Student("A",6);

 Classroom cm=**new** Classroom();

 cm.registerStudent(s1);

**Sample Output**

**Registered**

**NOTE:**

* You can make suitable function calls and use the **RUN CODE** button to check your **main()** method output.
* Make sure that all the strings in the return statement are case-sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q40

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Email:

data members:

String username

String domain

Email(String username, String domain): constructor **with** **public** visibility

**class** Validator:

method definitions:

checkEmail(Email email) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

signIn(Email email) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** EmailException:

method definitions:

EmailException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Email**

**-**define the **String**variable **username.**

**-**define the **String** variable **domain**

**-**define a **constructor** according to the above specifications.

Class **Validator**

**Implement the below methods for this class:**

**-String** **checkEmail(Email email) throws Exception:**

* Write a code to validate the criteria for getting the award.
* **throw an EmailException**if the **username** contains any special symbol with the message "**Invalid username**".
* **throw an EmailException**if the **domain is other than "outlook", "gmail", "hotmail", or "godaddy" with a**message "**Invalid domain**".
* If no above exception is found then return a string message "**Continue**".

**-String signIn(Email email) throws Exception:**

* Write a code to send an invite to the nominee.
* If **checkEmail()**method throws an **EmailException**then returns a message "**Failed**".(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* If no exception is found then return a message "**Signed in**".

**class EmailException extends Exception**

* Define **EmailException** class derived from Exception class

**Sample Input**

Email e=**new** Email("Shoyab@","gmail");

Validator v = **new** Validator();

String s = v.checkEmail(e);

String t = v.signIn(e);

s.toLowerCase();

t.toLowerCase();

**Sample Output**

**continue**

**signed** **in**

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q41

**Case Study**: There exists a monster who is going to fight the greatest warrior the world has ever seen. Before that, he needs to eat something. Given that, he eats only numbers. If anything other than numbers is found in his food he will throw the food and will not go for the war. **For example**, the food is like **"1234$56**". He will throw the food saying there exists a '$' which he considers as intolerable.

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Food:

data members:

String plate

int numberOfSoldiers

Food(String plate, int numberOfSoldiers): constructor **with** **public** visibility

**class** KhumbhKaran:

method definitions:

getReadyToFight(Food food) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

winTheWar(Food food) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** FoodException:

method definitions:

FoodException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Food**

**-**define the **String**variable **plate.**

**-**define the **int** variable **numberOfSoldiers**

**-**define a **constructor** according to the above specifications.

Class **KhumbhKaran**

**Implement the below methods for this class:**

**-String** **getReadyToFight(Food food) throws Exception:**

* Write a code to validate the criteria for going to fight.
* **throw a FoodException**if the **plate contains a special character**with a message "**Insect found in the food".**
* **throw a FoodException if**the plate contains an uppercase or lowercase character with a message "**Not edible".**Assuming that, either it will have special symbols or characters but not both simultaneously.
* If the **numberofSoldiers** is less than 1000 then throw a **FoodException**with the message "**Food was not sufficient**".
* If no above exception is found then return a string message "**Ready for the war**".

-**String winTheWar(Food food):**

* Write a code to send an invite to the nominee.
* If **getReadyToFight()** method throws a **FoodException** then returns a message "**No war**".(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* If no exception is found then return a message "**Here the battle starts**".

**Sample Input**

Food f = **new** Food("1234", 70000);

KhumbhKaran v = **new** KhumbhKaran();

String s = v.getReadyToFight(f);

String t = v.winTheWar(f);

s.toLowerCase();

t.toLowerCase();

**Sample Output**

ready for the war

here the battle starts

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q42

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Laptop:

data members:

int price

String details

Rating(String details, int price): constructor **with** **public** visibility

**class** Afford:

method definitions:

checkConfiguration(Laptop laptop) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

purchaseLaptop(Laptop laptop) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** LaptopException:

method definitions:

LaptopException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Laptop**

**-**define the **int**variable **price.**

**-**define the **String** variable **details**

**The string variable contains the RAM, HardDisk storage, and Type of hard-disk ie SSD or HDD separated by a '/'. For example, "8/512/HDD".**

**-**define a **constructor** according to the above specifications.

Class **Afford**

**Implement the below methods for this class:**

**-String** **checkConfiguration(Laptop laptop) throws Exception:**

* Write a code to validate the criteria for purchasing the laptop.
* **throw a LaptopException**if the **price** is greater than **70000**with the message "**Price too high**".
* **throw a LaptopException**if **RAM** is less than **8**with the message "**Minimum 8 RAM required**".
* **throw a LaptopException**if **HardDisk storage** is less than **256**with the message "**Minimum 256 space required**".
* **throw a LaptopException**if **HardDisk type**is **HDD**with the message "**SSD required**".
* If no above exception is found then return a string message "**Can be purchased**".

**-String purchaseLaptop(Laptop laptop):**

* Write a code to send an invite to the nominee.
* If **checkConfiguration**method throws a **LaptopException** then return a message "**Change configuration**".(Use try-catch block)
* If it throws any other exception then return a message "**other exception**".
* If no exception is found then return a message "**Perfect configuration**".

**Sample Input**

Laptop laptop=**new** Laptop("8/512/SSD", 58000);

Afford af=**new** Afford();

String s=af.checkConfiguration(laptop);

String t=af.purchaseLaptop(laptop);

s.toLowerCase();

t.toLowerCase();

**Sample Output**

can be purchased

perfect configuration

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q43

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class WalkingBoy:

int ﻿stepSize

int blockSize

visibility : public

WalkingBoy(int stepSize, int blockSize) : Constructor **with** **public** visibility

﻿method definition:

﻿targetHit(**String** platform) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

**class** BombBlast extends **Exception**:

﻿method definition:

﻿ BombBlast(**String** msg)

visibility: **public** ﻿

**Task**

**Class** **WalkingBoy**

-define all the data members as per the given specifications.

-define the constructor with public visibility.

**-Implement the below methods for this class:**

**-String** **targetHit(String platform) throws Exception:**

* Write a code that checks whether a **boy hit the target/bomb or not.**
* **platform** is a string that contains alphanumeric values.**divide the platform into N block of length blockSize.** For every block**check if the first stepSize character contains x in it**then throw **BombBlast**Exception with a message **"You hit the target".**
* Else **return "Win".**

**Class** **BombBlast extends Exception:**

-Define**BombBlast** class derived from the Exception class

**Example for Reference**

*stepSize = 2, blockSize = 3, platform = "1212121x212"*

*After dividing the platform into N blocks of blockSize = 3 -> ["121", "212", "1x2", "12"]*

*After consedering only stepSize for every block -> ["12", "21", "1x", "12"]*

*3rd block contains x. therefore it throw****BombBlast****exception.*

**Sample Input**

WalkingBoy boy = **new** WalkingBoy(2,3);

boy.targetHit("1212121x212");

**Sample Output**

**BombBlast** : You hit the target

**NOTE:**

* You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.
* stepSize is always less than equal to the blockSize not possibly for the last block.

**ALLOWED TECHNOLOGIES**

* Java 8

Q44

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Problem:

data members:

int credits

String type

Problem(int credits, String type): constructor **with** **public** visibility

**class** Checker:

method definitions:

checkProblem(Problem p) throws **Exception**:

**return** **type**: **String**

visibility: **public**

﻿

approveProblem(Problem p):

**return** **type**: **String**

visibility: **public**

﻿

**class** ProblemException:

method definitions:

ProblemException(**String** msg)

visibility: **public** ﻿

**Task**

Class **Problem**

**-**define the **int**variable **credits and string variable type.**

**-**define a **constructor** according to the above specifications.

Class **Checker**

**Implement the below methods for this class:**

**-String** **checkProblem(Problem p) throws Exception:**

* Write a code to validate the criteria for approving the problem.
* **throw a ProblemException**if credits of the problem are less than 10 with a message "**Insufficient credits".**
* **throw a ProblemException**if the type of the problem is **"String"** with a message "**String problem found".**
* **throw a ProblemException**if the type of the problem is **"Generic"** with a message "**Generic problem found".**
* **throw a ProblemException**if the type of the problem is **"I/O"** with a message "**I/O problem found".**
* else return "**Exception problem found**".

**-String approveProblem(Problem p):**

* Write a code that approves the problem.
* If **checkProblem**method throws a **ProblemException**then returns a message "**Not approved**".(Use try-catch block)
* If it throws any other exception then return a message "**Other exception**".
* If no exception is found then return a message "**Problem approved**".

**Sample Input**

Problem p=**new** Problem(41,"Exception");

Checker c=**new** Checker();

String s = c.checkProblem(p);

String t=c.approveProblem(p);

**Sample Output**

exception problem found

problem approved

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q45

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 Customer:

data member:

String name

long mobile

﻿ char key

visibility: public

﻿﻿ Customer(String name, long mobile, char key):constructor with public visibility

method definition:

﻿encrypt():

return : String﻿﻿

visibility : public

minCount():

return : long

﻿visibility : public

Task

**class Customer**

- define data members according to the above specifications

-define a constructor to the above specifications

-Implement the below methods for this class:

**-String encrypt():**

* ﻿Write a code to calculate encrypt the name of the customer.
* If the name already contains that key at any index independent of occurrence then returns "**Already Encrypted".**
* If the name does not contain a key at any index independent of occurrence then return change the name instance with the encrypted name and also return the encrypted name with the following rule -

1. It gets the **first digit (x) of the mobile number.**
2. Then it **inserts the key x times at the last of the name.**

**-long minCount():**

* ﻿Write a code that returns the minimum digit of the mobile number. For example - the minimum digit of the mobile number **"98979593**" is **3.**

**Sample Input**

Customer customer = **new** Customer("Doselect", 299999992, '#');

System.**out**.println(customer.encrypt());

System.**out**.println(customer.minCount());

**Sample Output**

Doselect##

﻿2

**Explanation**

The minimum digit of the mobile number (99999992) is 2; Therefore two "#" are inserted in the name.

**Note -**The output string can be different as the key has to be inserted at different indexes.

**NOTE:**

* You can make suitable function calls and use the **RUN CODE** button to check your **main()** method output.
* Make sure that all the strings in the return statement are case-sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q46

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Register:

﻿method definitions:

﻿checkCredentials(String email, String pass, String cpass) throws Exception:return type: String

visibility: public

﻿

class InvalidEmailException extends Exception:

﻿method definitions:

﻿InvalidEmailException(String msg)

visibility: public ﻿

class InvalidPasswordException extends Exception:

﻿method definitions:

﻿InvalidPasswordException(String msg)

visibility: public ﻿

class PasswordNotMatchException extends Exception:

﻿method definitions:

﻿ PasswordNotMatchException(String msg)

visibility: public ﻿

**Task**

class **Register**

**Implement the below methods for this class:**

**-String** **checkCredentials(String email, String pass, String cpass):**

* Write a code to validate credentials.
* Throw these exceptions **considering the conditions in the same sequence in which they are given** -

1. throw an **InvalidEmailException,** if the email does not contain **"@" and ".",**with the message **"Invalid Email".**
2. throw an **InvalidPasswordException**, if a **pass length is less than 6 characters,**with the message **"Invalid Password".**
3. throw a **PasswordNotMatchException**, if a**cpass is not equal to the valid pass,**with a message **"Password not match".**

* If the email contains **"@" and "."**in it and the pass is the same as cpass with a length greater than equal to 6 then return **"Registered"**.

*Refer the example for better understanding.*

*email1 = "myemail@email"*

*email2 = "myemail@email.com"*

*pass = "pass1234"*

*cpass = "pass123"*

*email1, pass and cpass gives****InvalidEmailException****as email validation fails and it has to be checked at first as per the given sequence.*

*email2, pass and cpass gives returns****Registered.***

class **InvalidEmailException**

-Define **InvalidEmailException** class derived from the Exception class

class **InvalidPasswordException**

-Define **InvalidPasswordException** class derived from the Exception class

class **PasswordNotMatchException**

-Define **PasswordNotMatchException** class derived from the Exception class

**Sample Input**

Register user = **new** Register();

user.checkCredentials("tushar@gmailcom","hiiiiii","hiiiiii");

**Sample Output**

**InvalidEmailException**: Invalid Email

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q47

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Customer:

data members:

String name

int numberOfRooms

Person(String name, int numberOfRooms): constructor **with** **public** visibility

**class** Hotel:

**data** members:

**List**<**String**> customerName

int roomsBooked = 0

method definitions:

checkIn(Customer customer):

**return** **type**: **String**

visibility: **public**

checkRooms(Customer customer) :

**return** **type** : **String**

visibility : **public**﻿﻿

Task

Class **Customer**

**-** define all the variables according to the above specifications.

-the **name**variable stores the names of all the guests separated by a comma.

-**numberOfRooms**defines the number of rooms needed by them.

**-** define a **constructor** according to the above specifications.

Class **Hotel**

**-** define all the variables according to the above specifications.

- define**roomsBooked with 0.**

**Assuming there are only 10 rooms in the hotel.**

**Implement the below methods for this class:**

**-String** **checkIn(Customer customer):**

* Write a code that checks if there are rooms available as per the requirement of the customer.
* If available, store the names of the guests in the ArrayList separately and return "**Checked in**".
* If rooms are not available then return "**No rooms available**".

**-String** **checkRooms(Customer customer):**

* Write a code that helps to find out whether the rooms required for the particular customers are available or not.
* If they are available then return "**Yes**" else return a "**No**".

**Sample Input**

Customer cust=**new** Customer("Doselect", 4);

Hotel hotel=**new** Hotel();

hotel.checkIn(cust);

**Sample Output**

Checked **in**

**NOTE:**

* You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.
* All the messages used in the return statements and messages are case-sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q48

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class RacingCar:

int ﻿speed

int distance

int petrolRate

visibility : public

RacingCar(int speed, int distance, int petrolRate) : Constructor **with** **public** visibility

﻿method definitions:

﻿validateCar(int time) throws **Exception**:

**return** **type**: int

visibility: **public**

﻿ calculateprice() :

**return** **type** : int

visibility : **public**

﻿

**class** LowSpeedException extends **Exception**:

﻿method definitions:

﻿ LowSpeedException(**String** msg)

visibility: **public**

**class** HighSpeedException extends **Exception**:

﻿method definitions:

﻿ HighSpeedException(**String** msg)

visibility: **public** ﻿

**Task**

**Class** **RacingCar**

-define all the data members as per the given specifications.

-define the constructor with public visibility.

**-Implement the below methods for this class:**

**-int** **validateCar(int time) throws Exception :**

* Write a code to validate the RacingCar.
* throw a **LowSpeedException,** if the calculated time(ceil value of (distance/time)) is greater than parameter timewith the message **"Upgrade Car".**
* throw a **HighSpeedException,** if the calculated time(ceil value of (distance/time)) is less than parameter timewith the message **"Accident Chances".**
* Else **return**the result calculated by **calculateprice.**

**-int calculateprice():**

* Write a code that returns the quantity of petrol need.
* The petrol quantity is defined as**Ceil value of distance/petrolRate.**

**Class** **LowSpeedException extends Exception**

-Define**LowSpeedException** class derived from the Exception class

**Class** **HighSpeedException extends Exception**

-Define**HighSpeedException** class derived from the Exception class

**Sample Input**

RacingCar car = **new** RacingCar(100,1000,11);

car.validateCar(50);

**Sample Output**

**Accident** Chances

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q49

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Build:

data members:

int length

int width

visibility : public

Build(int length, int width): constructor **with** **public** visibility

﻿ method definition:

builder(int blength, int bwidth, int **count**) throws ShortageException, TendorException:

**return** : **String**

visibility : **public**

﻿

**class** ShortageException extends **Exception**:

method definitions:

ShortageException(**String** msg)

visibility: **public**

**class** TendorException extends **Exception**:

method definitions:

TendorException(**String** msg)

visibility: **public**

**Class Build**

**-** define all the variables according to the above specifications.

**-** define a **constructor** according to the above specifications.

**Implement the below methods for this class:**

**-String** **builder(int blength, int bwidth,int count) throws ShortageException, TendorException:**

* Write a code that accepts the length, width, and count of the bricks available and return the result according to the mentioned scenarios below -

1. If the**brick length(blength) evenly divides the parameter length, brick width(bwidth) evenly divides the parameter width, and the number of bricks required to cover the total area(length\*breadth) is less than equal to the parameter count** then return **"Builder!!".**
2. If the**brick length(blength) evenly divides the parameter length, brick width(bwidth) evenly divides the parameter width, and the number of bricks required to cover the total area(length\*breadth) is greater than the parameter count** then throw the **ShortageException**with the message **"Need more bricks".**
3. If the**brick length(blength) does not evenly divides the parameter length or brick width(bwidth) does not evenly divide the parameter width, and the number of bricks required to cover the total area(length\*breadth) is less than equal to the parameter count** then throw the **TendorException**with the message **"Building dimension mismatched".**
4. If the**brick length(blength) does not evenly divides the parameter length or brick width(bwidth) does not evenly divide the parameter width, and the number of bricks required to cover the total area(length\*breadth) is greater than the parameter count** then throw the **ShortageException**with the message **"Need more bricks with dimension mismatched".**

**Class ShortageException**

- define **ShortageException** class derived from the Exception class.

**Class** **TendorException**

- define **TendorException** class derived from the Exception class.

**Sample Input**

Build build = **new** Build(100, 100);

build.builder(10,10,100);

build.builder(10,10,50);

**Sample Output**

Builder!!

**ShortageException: Need more bricks﻿**

NOTE:

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q50

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Song:

data members:

String name

String runTime

String singer

Song(String nm,String rt, String sng): constructor with public visibility

class PlayList:

data members:

List<Song> songs = new ArrayList<>();

method definitions:

playSong(String currentTime,Song song):

return type : String

visibility : public

﻿﻿

addSong(Song song):

return type: String

visibility: public﻿﻿

Task

Class **Song**

**-** define all the below-mentioned variables with the specified specifications -

1. name- A string that defines the name of the song.
2. runTime- A string that stores the runtime in minutes and second separated by ':' (For example=> 10:30).
3. singer- A string that contains the name of the singer.

**-** define a **constructor** according to the above specifications.

Class **PlayList**

**-** define the class according to the above specifications and

**Implement the below methods for this class:**

**-String** **playSong(String currentTime, Song song):**

* Write a code that validates that the passed **currentTime is less than the runTime of the song.**
* **If the conditions are satisfied then return "Song is being continued".**
* And if the condition is not satisfied then return a message "**Song replayed**".

**-String** **addSong(Song song):**

* Write a code that checks if the current song is present or not.
* If the song is present return "**Cannot be added**" else add to the ArrayList and return "**Added**"**.**

**Sample Input**

PlayList lis=**new** PlayList();

Song s1=**new** Song("Venom", "4:58","Eminem");

lis.addSong(s1);

**Sample Output**

**Cannot** be added

**NOTE:**

* You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.
* All the messages used in the return statements and messages are case-sensitive.
* The runTime and currentTime are provided in minutes and seconds.

**ALLOWED TECHNOLOGIES**

* Java 8

Q51

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class User:

data members:

String name

String city

String state

String pincode

String dob

visibility : private

User(String name, String city, String state, String pincode, String dob): constructor **with** **public** visibility

**define** getter **and** setter **with** **public** visibility

**class** EmployeeID:

**data** members:

**User** **user**

visibility : **public**

EmployeeID(**User** **user**): **constructor** **with** **public** visibility

﻿ method definition:

generateId():

**return** : **String**

visibility : **public**

pincodeValidator():

**return** : boolean

visibility : **public**

dobValidator():

**return** : boolean

visibility : **public**

Task

**Class User**

**-** define all the variables according to the above specifications.

**-** define a **constructor with getter and setter** according to the above specifications.

**Class EmployeeID**

**-** define all the variables according to the above specifications.

**-** define a **constructor** according to the above specifications.

**Implement the below methods for this class:**

**-String** **generateId():**

* Write a code that generates the employee id on the basis of the given condition below.

1. If the**pincodeValidator()** returns false than **return "NA".**
2. If **pincodeValidator() and dobValidator()** returns true than return the generated password. **Password = first three character of name) + (first character of city) + (first character of state) + (first 2 character of dob).**
3. If **pincodeValidator() returns true and** **dobValidator()** returns false than return the generated password. **Password = first three character of name) + (first character of city) + (first character of state).**

**-boolean** **pincodeValidator():**

* Write a code that **returns true** if the p**incode is of exactly 6 characters and all the characters are digits**otherwise **return false.**

**-boolean** **dobValidator():**

* Write a code that **returns true** if the **dob is of exactly 8 characters and there are two characters "/" in it**otherwise **return false.**

**Sample Input**

User user = **new** User("doselect","d","delhi","400001","22/04/2002");

EmployeeID employeeID = **new** EmployeeID(user);

employeeID.generateId()

**Sample Output**

**dosdd22**

NOTE:

* You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.
* All the messages used in the exception handling are case-sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q52

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Box:

data members:

int id

int volume

visibility : private

Box(int id, int volume): constructor **with** **public** visibility

**define** getter **and** setter **with** **public** visibility

**class** ContainerBox:

**data** members:

ArrayList<Box> boxes

visibility : publicmember definition:

addBox(Box box):

**return** void

visibility **public**

addWater(int **id**, int liter):

**return** : **String**

visibility : **public**

getBoxes(int volume, int comparator):

**return** : ArrayList<Box>

visibility : **public**

Task

**Class Box**

- define all the variables according to the above specifications.

**-** define a **constructor with getter and setter** according to the above specifications.

**Class ContainerBox**

**-** define all the variables according to the above specifications.

**Implement the below methods for this class:**

**-void** **addBox(Box box):**

* Write a code that adds the box parameter to the ArrayList (boxes)

**-String** **addWater(int id, int liter):**

* Write a code that adds fill the given amount of litter in the box according to the given conditions -

1. If the box with the given id is present in the boxes and the volume of the box is greater than equal to the given liter then decrease the volume of the box with the given liter and return "Water filled".
2. If the box with the given id is present in the boxes and the volume of the box is less than the given liter then return "Box full.
3. If there is no box with the given id then return "No Box found".

**-ArrayList<Box> getBoxes(int volume, int comparator):**

* Write a code that returns the ArrayList of Box on the basis of given conditions -

1. If the comparator value is 1 then return all the boxes that have a volume less than the given volume.
2. If the comparator value is 2 then return all the boxes that have a volume greater than the given volume.
3. Else return null.

**Sample Input**

Box box = **new** Box(1,200);

ContainerBox containerBox = **new** ContainerBox();

containerBox.addBox(box);

containerBox.addWater(1,100);

containerBox.getBoxes(200,4);

**Sample Output**

Water filled

null

**Explanation**

addBox method will add the box to the boxes list.

addWater will return "Water filled" and decrease the quantity of box by 100.

getBoxes will return null as a comparator is other than 1 and 2.

NOTE:

* You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.
* All the messages used in the exception handling are case-sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q53

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 Dish

data members:

dishName: String

Dish(String dishName): constructor **with** **public** visibility

**Define** getter setters **with** **public** visibility

toString() method has been implemented **for** you

**class** DishTest:

method definition:

addYummyToName(**List**<Dish> **list**, **String** s):

**return** **type**: **List**<Dish>

visibility: **public**

**count**(**List**<Dish> **list**):

**return** **type**: **Long**

visibility: **public**

**Task:**

**class Dish:**

**-**define the **String** variable **dishName**

**-**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 DishTest:**

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

* **List<Dish> addYummyToName(List<Dish> list):**Add Yummy to the dishName(refer to the sample output for the format), put it into a list and return the list
* **Long countI(List<Dish> list):**return the count of dishes having string **s**in name

**Sample Input**

DishTest dt = **new** DishTest();

**List**<Dish> **list** = **new** ArrayList<Dish>();

**list**.add(**new** Dish("Gazpacho"));

**list**.add(**new** Dish("Jamon"));

**list**.add(**new** Dish("Tortilla"));

**list**.add(**new** Dish("Churros"));

-------------------------------------------------------

dt.addYummyToName(**list**)

dt.count(**list**, "S")

**Sample Output**

[Dish{dishName='Yummy: Gazpacho'}, Dish{dishName='Yummy: Jamon'}, Dish{dishName='Yummy: Tortilla'}, Dish{dishName='Yummy: Churros'}]

---------------------------------------------------------

0

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q54

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 Kingdom:

﻿data members:

﻿String life;

String nonLife;

int lifeSpan;

﻿

Kingdom(String life, String nonLife, int lifeSpan): constructor **with** **public** visibility

﻿ **Define** getter **with** **public** visibility

﻿ toString(): has been implemented **for** you

**class** KingdomClassification:

﻿method definition:

﻿ getKingdom(**List**<Kingdom> **list**):

﻿ **return** **type**: **List**<**String**>

visibility: **public**

findNameWithValidity(**List**<Kingdom> **list**, **String** **name**, int lifeSpan):

﻿ **return** **type**: Kingdom

visibility: **public**

**Task:**

**class Kingdom:**

**-**define class **Kingdom** according to the above specifications

class **KingdomClassification:**

**Implement the below method for this class**:

* **List<Strig>getKingdom(List<Kingdom> list):**fetch and return the kingdom classified as nonLife from the list
* **Kingdom findNameWithValidity(List<Kingdom> list, String name, int lifeSpan):**return the kingdom details with the given name and lifeSpan if found else return null

*Refer sample output for clarity*

**Sample Input**

KingdomClassification k = **new** KingdomClassification();

**List**<Kingdom> **list** = **new** ArrayList<Kingdom>();

**list**.add(**new** Kingdom("Protista", "Regnum", 3));

**list**.add(**new** Kingdom("Plantae", "Lapideum", 5));

---------------------------------------------------------

k.getKingdom(**list**)

k.findNameWithValidity(**list**, "Lapideum", 5)

**Sample Output**

[Protista, Plantae]

----------------------------------------------------------

Kingdom{life='Plantae', nonLife='Lapideum', lifeSpan=5}

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q55

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 Fish:

﻿data members:

﻿ String species;

int weight;

﻿

Fish(String species, int weight): constructor **with** **public** visibility

﻿ **Define** getter **with** **public** visibility

﻿ toString(): has been implemented **for** you

**class** FishClass:

﻿method definition:

﻿ getFishWithLowestWeight(**List**<Fish> **list**)

﻿ **return** **type**: **List**<Fish>

visibility: **public**

mapFishWeight(**List**<Fish> **list**):

﻿ **return** **type**: **List**<Integer>

visibility: **public**

**Task:**

**class Fish:**

**-**define class **Fish** according to the above specifications

class **FishClass:**

**Implement the below method for this class**:

* **List<Fish>getFishWithLowestWeight(List<Fish> list):**fetch and return the the fish whose weight is lowest amongst all
* **List<Integer>mapFishWeight(List<Fish> list):**fetch and return the weight of the fishes from the list

*Refer sample output for clarity*

**Sample Input**

FishClass f = **new** FishClass();

**List**<Fish> **list** = **new** ArrayList<Fish>();

**list**.add(**new** Fish("BlobFish", 20));

**list**.add(**new** Fish("Guppy", 15));

**list**.add(**new** Fish("Oscar", 60));

**list**.add(**new** Fish("GoldFish", 5));

--------------------------------------------------------

f.getFishWithLowestWeight(**list**)

f.mapFishWeight(**list**)

**Sample Output**

[Fish{species='GoldFish', weight=5}]

----------------------------------------------------------

[20, 15, 60, 5]

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q55

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 Student:

﻿data members:

﻿ name: String

visibility: public

﻿ score: int

visibility: private

﻿

Student(String name, int score): constructor **with** **public** visibility

﻿ **Define** getter **with** **public** visibility

﻿ toString(): has been implemented **for** you

**class** ReportCard:

﻿method definition:

﻿ mapStudentName(**List**<Student> **list**):

﻿ **return** **type**: **List**<**String**>

visibility: **public**

scoreQuery(**List**<Student> **list**):

﻿ **return** **type**: **List**<Student>

visibility: **public**

**Task:**

**class Student:**

**-**define class **Student** according to the above specifications

class **ReportCard:**

**Implement the below method for this class**:

* **List<String>mapStudentName(List<Student> list):**fetch and return the student name from the list
* **List<Student>scoreQuery(List<Student> list):**filter the score from the list greater than **40** and less than equal to **100**, put it into a list and return the desired list

*Refer sample output for clarity*

**Sample Input**

ReportCard r = **new** ReportCard();

**List**<Student> **list** = **new** ArrayList<Student>();

**list**.add(**new** Student("Sarah", 70));

**list**.add(**new** Student("John", 99));

**list**.add(**new** Student("Jenny", 39));

**list**.add(**new** Student("Harry", 22));

---------------------------------------------------------

 r.mapStudentName(**list**)

 r.scoreQuery(**list**)

**Sample Output**

[Sarah, John, Jenny, Harry]

----------------------------------------------------------

[Student{name='Sarah', score=70}, Student{name='John', score=99}]

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q56

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** **Implementation**:

   method definitons:

    count(List<String> products):

**return** type: **int**

      visibility: **public**

    match(List<String> products):

**return** type: String

      visibility: **public**

**Task:**

**class Implementation**

-Implement the below methods for this class in using **Stream Api** methods

* **int count(List<String> products):** return the **count** of components in**products**
* **String match(List<String> products)**: Method to **check** if there is letter**z**present in any element of Arraylist
* Print **"Great Job that sentence does not contain z"**if elements in Arraylist doesn't contain**z**
* Print**"Cant fool the system, that sentence contain z"**if elements in Arraylist contain **z**

*Implement using****Lambda expressions****.*

**Sample Input**

 Implementation i = **new** Implementation();

**List**<String> **list** = **new** ArrayList<String>();

**list**.add("Gadgets");

**list**.add("FoodItem");

**list**.add("Pets");

**list**.add("Appliances");

-----------------------------------------------------------

i.count(**list**)

i.match(**list**)

**Sample Output**

4

-----------------------------------------------------------

Great Job that sentence does **not** contain z

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8

Q57

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 Lecturer:

﻿data members:

String lecturerFirstName

String lecturerLastName

int salary

visibility: private

﻿

Lecturer(String lecturerFirstName, String lecturerLastName, int salary): constructor **with** **public** visibility

Implement getter **and** setter methods **for** this **class** **with** **public** visibility

﻿ toString(): has been implemented **for** you

**class** Implementation:

﻿

﻿method definition:

﻿ getFirstName(**List**<Lecturer> **list**):

**return** **type**: **List**<**String**>

﻿ visibility: **public**

getMaxSalary(**List**<Lecturer> lecturerList)

﻿ **return** **type**: **double**

visibility: **public**

**Task:**

**class Lecturer:**

**-**define class **Lecturer** according to the above specifications

class **Implementation:**

**-Implement the below method for this class**:

* **List<String> getFirstName(List<Lecturer> list):**fetch andreturn the **first name** of all the lecturer from the list
* **double getMaxSalary(List<Lecturer> list): f**etch the salary from the list which is **maximum**

***Refer sample output for clarity***

**Sample Input**

Implementation i = **new** Implementation();

**List**<Lecturer> **list** = **new** ArrayList<Lecturer>();

**list**.add(**new** Lecturer("Alan", "D'costa", 500000));

**list**.add(**new** Lecturer("Tom", "Sal", 1000000));

**list**.add(**new** Lecturer("John", "Mirra", 360000));

----------------------------------------------------------

i.getFirstName(**list**)

i.getMaxSalary(**list**)

**Sample Output**

[Alan, Tom, John]

**-----------------------------------------------------------**

1000000.0

**NOTE**

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

**ALLOWED TECHNOLOGIES**

* Java 8
* Tag:
* Stream API Arraylist

Q58

Complete the classes using the Specifications given below. Consider default visibility of classes, data fields, and methods unless mentioned otherwise.

**Specifications**

class definitions:﻿

class Laptop:

﻿ data members:

String name

String model

String date

Laptop(String name, String model, String date): constructor **with** **public** visibility

**class** Purchase:

**data** members:

laptop laptop

method definitions:

﻿purchaseLaptop(**String** **name**, **String** **model**,**String** date):

**return** **type**: **String**

visibility: **public**

﻿

﻿ findGeneration(**String** processor)

**return** **type** : **String**

visibility : **public**

class **Laptop**

**-** define all the variables according to the above specifications.

**-** define a **constructor** according to the above specifications.

Class **Purchase**

**-** define all the variables according to the above specifications.

**Implement the below methods for this class:**

**-String** **purchaseLaptop(String name, String model, String date):**

* Check the following conditions -

1. The name must be of two words. The first word denotes the name of the brand [**"Dell", "Acer", "HP"**] and the second word is the number of processors [**"Single", "Double", "Quad", "Octa"**]. If the company name is not among the mentioned then return "**Invalid brand name**". If the processor is any other than the mentioned then return "**Invalid** **processor**".
2. model name length must be equal to 9 with 3 uppercase letters, 3 lowercase letters, 2 digits, and 2 special characters. If this condition is not fulfilled return "**Invalid model number**".

* If the above all conditions are satisfied then initialize the laptop variable with the above-mentioned specifications, return "**Laptop purchased**".

**-String findGeneration(String processor):**

* If the number of processors is "**Octa**" return "**Gen 11**".
* If the number of processors is "**Quad**" return "**Gen 10**".
* If the processor is "**Double**" return **"Gen 9"**.
* Else return "**Lower than 9**".

**Sample Input**

Laptop l=**new** Laptop("Acer Quad","@#","12/12/2018");

Purchase p=**new** Purchase();

p.purchaseLaptop("Dell Quad","@#","12/12/2018");

**Sample Output**

**Invalid** model number

**NOTE:**

* You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.
* All the messages used in the return statements and messages are case-sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q59

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 Student:

data members:

String name

String branch

int age

int score

visibility: public

﻿

﻿ Student(String name, String branch, int age,int score ): constructor **with** **public** visibility

**class** Company:

**data** members:

**String** nameString typeboolean drive

int cutoff

visibility: **public**

﻿

﻿ Company(**String** **name**, **String** **type**, int cutoff,boolean drive): **constructor** **with** **public** visibility﻿

**class** Recruitment:

**data** members :

ArrayList<Company> companies

Student student = null

visibility : **public**

﻿

method definition:

addStudent(**String** **name**, **String** branch, int age, int score):

**return** : **String**

visibility **public**

addCompany(Company company):

**return** : **String**

visibility **public**﻿

eligibleCompanies(Student student):

**return** : **String**

visibility : **public**

passedOnlineTest(int score) :

**return**: int

laggingScores(int **value**):

**return** : int

visibility : **public**

**class Student**

- define data members according to the above specifications

-define a constructor and getters setters according to the above specifications

**class Company**

- define data members according to the above specifications

-define a constructor and getters setters according to the above specifications

**class Recruitment**

- initialize student as null initially.

-Implement the below methods for this class:

**-String addCompany(Company company):**

* ﻿Write a code to add a given room object to the 'companies' ArrayList.
* Add the company object to the companies list and return **"Added".**

**-String addStudent(String name, String branch, int age, int score):**

* ﻿Write a code that removes extra spaces from the student's name and updates the same name.. There should not be two spaces in a consecutive way.
* Check the branch of the student is "CSE", "IT", "ME", "CE", "EEE" or "ECE"..
* Consider the following condition to implement the method -

1. return "Invalid branch" if the branch is not in the above-mentioned list.
2. return "Student added" if all the conditions are satisfied.

**-String addCompany(Company company):**

* ﻿Write a code via which we can check if the company is already present in the list or not.
* If the company is already present check whether the drive is true or false.
* If the drive is true return "Already Recruited" else update the details of the company and return "Details modified".
* If the company is not present in the list then add the company and return "Recruiting".

**-String eligibleCompanies(Student student):**

* ﻿Check for the branch of student and match with the type of each company. Return the name of the companies separated by a comma that has the same type as the student's branch.
* If no company matches the criteria then return "Not eligible in any company".

**-int passedOnlineTest(int score):**

* ﻿Write a code to check for how many companies the student has scored more than the cutoff of the given company. The passed parameter represents the score of the student.
* Return the number of companies passed.

**-int laggingScore(int value,Company c):**

* ﻿Write a code to calculate the marks needed to pass an exam for the given company c.
* Return the marks difference.

**Sample Input**

Company c1=**new** Company("Doselect","CSE",23,false);

Recruitment rec=**new** Recruitment();

rec.addCompany(c1)

**Sample Output**

**Recruiting**

**NOTE:**

* You can make suitable function calls and use the **RUN CODE** button to check your **main()** method output.
* Make sure that all the strings in the return statement are case-sensitive.

**ALLOWED TECHNOLOGIES**

* Java 8

Q60

Alan has recently learned about strings in his programming classes. He decided to create some interesting strings using the concepts.

Help Alan!

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** **Implementation**:

 method definiton:

  splitAddAndReturnCharacter(String str)

visibility: **public**

**return** type: **char**

**Task:**

class **Implementation**

-Implement the below method for this class

* **char splitAddAndReturnCharacter(String str):** Sum all the character values i.e. **‘a’ = 1, ‘b’ = 2, ‘c’ = 3, …, ‘z’ = 26** and return the character corresponding to the sum value. If it exceeds 26 then take sum % 26.

Note: **string str** consisting of lowercase English alphabets only.

**Sample Input:**

**java**

**Sample Output**

**h**

**NOTE:**

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

**ALLOWED TECHNOLOGIES**

* Java 8