Rajalakshmi Engineering College

Name: NISHANTH ELANGO RAJAN
Email: 241901075@rajalakshmi.edu.in
Roll no: 241901075
Phone: 9444909050
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 8_Q1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Write a program to validate the email address and display suitable exceptions if there is any mistake.

Create 3 custom exception classes as below

DotExceptionAtTheRateExceptionDomainException

A typical email address should have a " . " character, and a "@" character, and also the domain name should be valid. Valid domain names for practice be 'in', 'com', 'net', or 'biz'.

Display Invalid Dot usage, Invalid @ usage, or Invalid Domain message based on email id.

Get the email address from the user, validate the email by checking the

above-mentioned criteria, and print the validity status of the input email address.

Input Format

The first line of input contains the email to be validated.

Output Format

The output prints a Valid email address or an Invalid email address along with the suitable exception

If email ends with . or contains not exactly one . after @, it throws:

DotException: Invalid Dot usage

Invalid email address

If @ appears not exactly once, it throws:

AtTheRateException: Invalid @ usage

Invalid email address

If the part after the last dot is not among accepted domains:

DomainException: Invalid Domain

Invalid email address

If all conditions satisfied then print:

Valid email address

Refer to the sample input and output for format specifications.

Sample Test Case

Input: sample@gmail.com

Output: Valid email address

Answer

```
import java.util.Scanner;
class DomainException extends Exception {
    String expDescription;
    DomainException(String expDescription) {
        super(expDescription);
    }
 }
class DotException extends Exception {
    String expDescription;
    DotException(String expDescription) {
        super(expDescription);
    }
 }
class AtTheRateException extends Exception {
    String expDescription;
    AtTheRateException(String expDescription) {
        super(expDescription);
    }
 }
class EmailValidationMain {
   public static void main(String[] args) {
      Scanner myObj = new Scanner(System.in);
      String email = myObj.next();
      boolean checkEndDot  = false;
      checkEndDot = email.endsWith(".");
      int indexOfAt = email.indexOf('@');
      int lastIndexOfAt = email.lastIndexOf('.');
      int countOfAt = 0;
      for (int i = 0; i < email.length(); i++)
      {
         if(email.charAt(i)=='@')
            countOfAt++;
```

```
        }
        String buffering = email.substring(email.indexOf('@')+1, email.length());
        int len = buffering.length();
        int countOfDotAfterAt = 0;
        for (int i=0; i < len; i++) {
           if(buffering.charAt(i)=='.')
              countOfDotAfterAt++; }
        String userName = email.substring(0, email.indexOf('@'));
        String domainName = email.substring(email.indexOf('.')+1, email.length());
        int domainCheck=0;
        if((domainName.equals("in")) || (domainName.equals("com")) ||
(domainName.equals("net")) || (domainName.equals("biz")))
          domainCheck=1;
        try {
          if((checkEndDot) || (countOfDotAfterAt!=1)) {
            throw new DotException("Invalid Dot usage");
          }
          if(countOfAt!=1) {
            throw new AtTheRateException("Invalid @ usage");
          }
          if(domainCheck!=1) {
            throw new DomainException("Invalid Domain");
          }
        }catch(DotException e) {
          System.out.println(e);
        }catch(AtTheRateException e) {
          System.out.println(e);
        }catch(DomainException e) {
          System.out.println(e);
        }
        if ((countOfAt==1) && (userName.endsWith(".")==false)  &&
(domainCheck==1) && (countOfDotAfterAt ==1) &&((indexOfAt+3) <=
(lastIndexOfAt) && !checkEndDot)) {
          System.out.println("Valid email address");
        }
         else {
           System.out.println("Invalid email address");
         }
        myObj.close();
    }
}
```

Status : Correct

Marks : 10/10

Rajalakshmi Engineering College

Name: NISHANTH ELANGO RAJAN
Email: 241901075@rajalakshmi.edu.in
Roll no: 241901075
Phone: 9444909050
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 8_Q2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Elsa, a busy professional, is using a scheduling application to plan her meetings efficiently. The application requires users to input meeting durations in minutes, ensuring that the duration is a positive integer and does not exceed 240 minutes (4 hours). Elsa needs a program to assist her in scheduling meetings securely with proper exception handling.

Create a Java class named ElsaMeetingScheduler.Implement a custom exception:InvalidDurationException for invalid meeting duration entries.Implement the main method to interactively take user input for a meeting duration.Implement the validateMeetingDuration method to validate the meeting duration based on the specified rules and throw a custom exception if the validation fails.Print appropriate success or error messages based on the meeting duration.

Implement a custom exception, InvalidDurationException, to handle cases where the entered meeting duration does not meet the specified criteria.

Input Format

The input consists of an integer value 'n', representing the meeting duration.

Output Format

The output is displayed in the following format:

If the entered meeting duration meets the specified criteria, the program outputs

"Meeting scheduled successfully!"

If the entered meeting duration is invalid, the program outputs an error message indicating the issue.

"Error: Invalid meeting duration. Please enter a positive integer not exceeding 240 minutes (4 hours)."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 120
Output: Meeting scheduled successfully!

Answer

```
import java.util.Scanner;
class InvalidDurationException extends Exception {
   public InvalidDurationException(String message) {
      super(message);
   }
}
class MeetingValidator {
   public void validateMeetingDuration(int duration) throws
InvalidDurationException {
      if (duration <= 0 || duration > 240) {
         throw new InvalidDurationException(
            "Invalid meeting duration. Please enter a positive integer not exceeding
```

```
240 minutes (4 hours)."
            );
        }
    }
}
class MeetingScheduler {
    private MeetingValidator validator = new MeetingValidator();
    public void scheduleMeeting() {
        Scanner scanner = new Scanner(System.in);
        try {
            int meetingDuration = scanner.nextInt();
            validator.validateMeetingDuration(meetingDuration);
            System.out.println("Meeting scheduled successfully!");
        } catch (InvalidDurationException | java.util.InputMismatchException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
public class Main {
    public static void main(String[] args) {
        MeetingScheduler scheduler = new MeetingScheduler();
        scheduler.scheduleMeeting();
    }
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: NISHANTH ELANGO RAJAN
Email: 241901075@rajalakshmi.edu.in
Roll no: 241901075
Phone: 9444909050
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 8_Q3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

In a user registration system, there is a requirement to implement a username validation module. Users attempting to register must adhere to specific criteria for their usernames to be considered valid.

Your task is to develop a program that takes user input for a desired username and validates it according to the following rules:

The username must not contain any spaces.The username must be at least 5 characters long.

Implement a custom exception, InvalidUsernameException, to handle cases where the entered username does not meet the specified criteria.

Input Format

The input consists of a string S, representing the desired username.

Output Format

If the username is valid, print "Username is valid: [S]".

If the username is invalid:

1. If the username is short, print "Invalid Username: Username must be at least 5 characters long"
2. If the username contains spaces, print "Invalid Username: Username cannot contain spaces"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: John

Output: Invalid Username: Username must be at least 5 characters long

Answer

```
import java.util.Scanner;
class InvalidUsernameException extends Exception {
    public InvalidUsernameException(String message) {
        super(message);
    }
}
class UsernameValidator {
    public static void validateUsername(String username) throws
InvalidUsernameException {
        if (username.contains(" ")) {
            throw new InvalidUsernameException("Username cannot contain
spaces");
        }
        if (username.length() < 5) {
            throw new InvalidUsernameException("Username must be at least 5
characters long");
        }
    }
}
public class Main {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String username = scanner.nextLine();
        try {
            UsernameValidator.validateUsername(username);
            System.out.println("Username is valid: " + username);
        } catch (InvalidUsernameException e) {
            System.out.println("Invalid Username: " + e.getMessage());
        }
    }
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: NISHANTH ELANGO RAJAN
Email: 241901075@rajalakshmi.edu.in
Roll no: 241901075
Phone: 9444909050
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 8_Q4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

A local municipality is implementing an online voting system for a community event and wants to ensure that only eligible voters (those aged 18 or older) can participate.

Your task is to develop a program that validates the age of individuals attempting to vote online. If the user's age is below 18, the program should throw a custom exception, InvalidAgeException, preventing them from casting their vote. If the input is invalid, catch the appropriate InputMismatchException and print the in-built exception message.

Input Format

The input consists of an integer representing the age.

Output Format

If the age is 18 or older, print "Eligible to vote"

If the age is below 18, print "Exception occurred: InvalidAgeException: Age is not valid to vote"

If there is any other type of exception, print "An error occurred: " followed by the in-built exception message.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 20
Output: Eligible to vote

Answer

```
import java.util.Scanner;
class Main {
    static void validate(int age) throws InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age is not valid to vote");
        } else {
            System.out.println("Eligible to vote");
        }
    }
    public static void main(String args[]) {
        Scanner scanner = new Scanner(System.in);
        try {
            int userAge = scanner.nextInt();
            validate(userAge);
        } catch (InvalidAgeException ex) {
            System.out.println("Exception occurred: " + ex);
        } catch (Exception ex) {
            System.out.println("An error occurred: " + ex);
        } finally {
            scanner.close();
        }
    }
}
```

```
class InvalidAgeException extends Exception {
    public InvalidAgeException(String str) {
        super(str);
    }
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: NISHANTH ELANGO RAJAN
Email: 241901075@rajalakshmi.edu.in
Roll no: 241901075
Phone: 9444909050
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 8_Q5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

In a file management system, users are required to provide a valid file name when creating new files. The system enforces specific rules for file names to maintain consistency and avoid potential issues. Your task is to implement a Java program named FileNameValidator that takes user input for a file name and validates it according to the specified rules.

Rules for Valid File Name:

The file name must consist of alphanumeric characters (letters and digits) only.The file name must have a minimum length of 3 characters.

Implement a custom exception, FileNameValidator, to handle cases where the entered filename does not meet the specified criteria.

Input Format

The input consists of a string S, representing the desired filename.

Output Format

The output is displayed in the following format:

If the entered file name meets the specified criteria, the program outputs

"Valid file name"

If the entered file name does not meet the criteria and triggers the InvalidFileNameException, the program outputs

"Error: Invalid file name. It must be alphanumeric and have a minimum length of 3 characters."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: myfile123

Output: Valid file name

Answer

```
import java.util.Scanner;
class InvalidFileNameException extends Exception {
    public InvalidFileNameException(String message) {
        super(message);
    }
}
class FileNameValidator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            String fileName = scanner.nextLine();
            validateFileName(fileName);
            System.out.println("Valid file name");
        } catch (InvalidFileNameException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
```

```
        }
    }
    private static void validateFileName(String fileName) throws
InvalidFileNameException {
        if (!fileName.matches("^[a-zA-Z0-9]{3,}$")) {
            throw new InvalidFileNameException("Invalid file name. It must be
alphanumeric and have a minimum length of 3 characters.");
        }
    }
}
```

Status : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: NISHANTH ELANGO RAJAN
Email: 241901075@rajalakshmi.edu.in
Roll no: 241901075
Phone: 9444909050
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 8_CY

Attempt : 1
Total Mark : 40
Marks Obtained : 40

Section 1 : Coding

1. Problem Statement

Tim was tasked with creating a user profile system that validates the user's date of birth input. The system should throw a custom exception, InvalidDateOfBirthException, if the date is not in the specified format "dd-mm-yyyy" or if it represents an invalid calendar date.

The main method takes user input, validates the date of birth, and prints whether it is valid or not.

Input Format

The input consists of a string, representing the date of birth of the user.

Output Format

The output displays one of the following results:

If the entered date of birth is valid according to the specified format, the program prints:

"[Date] is a valid date of birth"

If the entered date of birth is not valid according to the specified format, the program prints:

"Invalid date: [Date]"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 01-01-2000

Output: 01-01-2000 is a valid date of birth

Answer

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;
class InvalidDateOfBirthException extends Exception {
    public InvalidDateOfBirthException(String message) {
        super(message);
    }
}
class UserProfileSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String userInput = "";
        try {
            userInput = scanner.nextLine();
            validateDateOfBirth(userInput);
            System.out.println(userInput + " is a valid date of birth");
        } catch (InvalidDateOfBirthException e) {
            System.out.println(e.getMessage() + ": " + userInput);
        } finally {
            scanner.close();
```

```
    }
  }
   private static void validateDateOfBirth(String userInput) throws
InvalidDateOfBirthException {
      SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
      dateFormat.setLenient(false);
      try {
         Date dob = dateFormat.parse(userInput);
      } catch (ParseException e) {
         throw new InvalidDateOfBirthException("Invalid date");
      }
   }
}
```

Status : Correct **Marks : 10/10**

2. Problem Statement

Camila, a user of a social media platform, is looking to change her password to enhance account security. The platform enforces specific rules for password strength to ensure the safety of user accounts. Camila needs a program that prompts her to enter a new password and throws custom exceptions based on the strength of the password.

Password Strength Criteria:

Weak Password:

Length less than 8 characters.Medium Password:

Length 8 or more characters.Missing a mix of uppercase letters, lowercase letters, and digits.

Implement a custom exception, to assist Camila in changing her password securely. The program should interactively take user input for a new password, categorize its strength, and handle custom exceptions (WeakPasswordException and MediumPasswordException) if the password fails to meet the specified criteria.

Input Format

The input consists of a string s, representing the new password.

Output Format

The output is displayed in the following format:

If the entered password meets the strength criteria, the program outputs

"Password changed successfully!"

If the entered password is weak, the program outputs

"Error: Weak password. It must be at least 8 characters long."

If the entered password is of medium strength, the program outputs

"Error: Medium password. It must include a mix of uppercase letters, lowercase letters, and digits."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: ComplexP@ss1

Output: Password changed successfully!

Answer

```
import java.util.Scanner;
class WeakPasswordException extends Exception {
    public WeakPasswordException(String message) {
        super(message);
    }
}
class MediumPasswordException extends Exception {
    public MediumPasswordException(String message) {
        super(message);
    }
}
class PasswordChangeSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
```

```
            String newPassword = scanner.nextLine();
            categorizePassword(newPassword);
            System.out.println("Password changed successfully!");
        } catch (WeakPasswordException | MediumPasswordException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
    private static void categorizePassword(String newPassword) throws
WeakPasswordException, MediumPasswordException {
        if (newPassword.length() < 8) {
            throw new WeakPasswordException("Weak password. It must be at least
8 characters long.");
        } else if (!containsUppercase(newPassword) || !
containsLowercase(newPassword) || !containsDigit(newPassword)) {
            throw new MediumPasswordException("Medium password. It must
include a mix of uppercase letters, lowercase letters, and digits.");
        }
    }
    private static boolean containsUppercase(String password) {
        return !password.equals(password.toLowerCase());
    }
    private static boolean containsLowercase(String password) {
        return !password.equals(password.toUpperCase());
    }
    private static boolean containsDigit(String password) {
        for (char c : password.toCharArray()) {
            if (Character.isDigit(c)) {
                return true;
            }
        }
        return false;
    }
}
```

Status : Correct **Marks : 10/10**

3. Problem Statement

A company is developing a user registration system that requires users to

provide valid email addresses. The development team is implementing an EmailValidator program to ensure that the entered email addresses meet certain criteria using exception handling.

The email address must contain the "@" symbol.The email address must consist of a non-empty username(before "@" symbol) and a non-empty domain(after "'@" symbol).The domain part of the email address must contain at least one period (".").The email address must not contain leading or trailing spaces.

Implement a custom exception, InvalidEmailException, to fulfill the company's requirements and validate it according to the specified rules.

Input Format

The input consists of a string value 's', which represents the email address.

Output Format

The output is displayed in the following format:

If the entered email address is valid according to the specified rules, the program prints:

"Email address is valid!"

If the entered email address misses the username or domain part or misses "@" symbol or has two or more "@" symbols or misses '.' in the domain part it outputs:

"Error: Invalid email format."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: johndoe@example.com
Output: Email address is valid!

Answer

```
import java.util.Scanner;
```

```
class EmailValidator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            String emailAddress = scanner.nextLine();
            validateEmailAddress(emailAddress);
            System.out.println("Email address is valid!");
        } catch (InvalidEmailException | java.util.InputMismatchException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
    private static void validateEmailAddress(String emailAddress) throws
InvalidEmailException {
        if (!emailAddress.contains("@")) {
            throw new InvalidEmailException("Invalid email format. ");
        }
        String[] parts = emailAddress.split("@");
        if (parts.length != 2 || parts[0].isEmpty() || parts[1].isEmpty() || !
parts[1].contains(".")) {
            throw new InvalidEmailException("Invalid email format. ");
        }
    }
}
class InvalidEmailException extends Exception {
    public InvalidEmailException(String message) {
        super(message);
    }
}
```

Status : Correct **Marks : 10/10**

4. Problem Statement

Theo is trying to update his payment information on a subscription-based streaming service. To proceed, the system requires Theo to provide a valid credit card number consisting of 16 digits. However, Theo wants to make sure that the credit card number he enters meets the specified criteria with proper exception handling.

The credit card number must consist of exactly 16 digits.If the entered credit card number does not meet the specified criteria, the program should throw a custom exception, InvalidCreditCardException, and provide Theo with specific error messages:If the length of the credit card number is not 16 digits, the exception message should be: "Invalid credit card number length."If the credit card number contains non-numeric characters, the exception message should be: "Invalid credit card number format."

Implement a custom exception, InvalidCreditCardException, to fulfill Theo's requirements and keep his payment information secure.

Input Format

The input consists of a string value 's', consisting of the 16-digit credit card number.

Output Format

The output is displayed in the following format:

If the entered credit card number is valid, the program should output a success message:

"Payment information updated successfully!"

If the entered credit card has more than 16 digits or less than 16 digits it displays

"Error: Invalid credit card number length."

If the entered 16-digit credit card has non-integers it displays

"Error: Invalid credit card number format."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1234567890123456
Output: Payment information updated successfully!

Answer

```
import java.util.Scanner;
class InvalidCreditCardException extends Exception {
    public InvalidCreditCardException(String message) {
        super(message);
    }
}
class CreditCardValidator {
    public void validateCreditCardNumber(String creditCardNumber) throws
InvalidCreditCardException {
        if (!creditCardNumber.matches("^\\d{16}$")) {
            if (creditCardNumber.length() != 16) {
                throw new InvalidCreditCardException("Invalid credit card number
length.");
            } else {
                throw new InvalidCreditCardException("Invalid credit card number
format.");
            }
        }
    }
}
class CreditCardUpdater {
    private CreditCardValidator validator = new CreditCardValidator();
    public void updateCreditCard() {
        Scanner scanner = new Scanner(System.in);
        try {
            String creditCardNumber = scanner.nextLine();

            validator.validateCreditCardNumber(creditCardNumber);

            System.out.println("Payment information updated successfully!");
        } catch (InvalidCreditCardException | java.util.InputMismatchException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
public class Main {
    public static void main(String[] args) {
        CreditCardUpdater updater = new CreditCardUpdater();
        updater.updateCreditCard();
```

```
    }
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: NISHANTH ELANGO RAJAN
Email: 241901075@rajalakshmi.edu.in
Roll no: 241901075
Phone: 9444909050
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 8_MCQ

Attempt : 1
Total Mark : 15
Marks Obtained : 15

Section 1 : MCQ

1. what is the output of the following code?

```
class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}

class Test {
    public static void main(String[] args) {
        try {
            throw new MyException("Error occurred");
        } catch (MyException e) {
            System.out.println(e);
        }
    }
```

```
}
```

Answer

MyException: Error occurred

Status : Correct ***Marks : 1/1***

2. What will be the output for the following code?

```
class InvalidUsernameException extends Exception {
   public InvalidUsernameException(String message) {
      super(message);
   }
}

class Test {
   public static void main(String[] args) {
      try {
         String username = "abc";
         if (username.length() < 5) {
            throw new InvalidUsernameException("Username must be at
least 5 characters long");
         }
      } catch (InvalidUsernameException e) {
         System.out.println(e.getMessage());
      }
   }
}
```

Answer

Username must be at least 5 characters long

Status : Correct ***Marks : 1/1***

3. What will happen if a checked custom exception is thrown inside a method without being caught or declared?

Answer

Compilation Error

Status : Correct **Marks : 1/1**

4. What will be the output for the following code?

```
import java.io.*;

class NegativeAgeException extends Exception {
   public NegativeAgeException(String message) {
      super(message);
   }
}

class Test {
   public static void main(String[] args) {
      try {
         int age = -5;
         if (age < 0) {
            throw new NegativeAgeException("Age cannot be negative");
         }
      } catch (NegativeAgeException e) {
         System.out.println(e.getMessage());
      }
   }
}
```

Answer

Age cannot be negative

Status : Correct **Marks : 1/1**

5. Which of the following is true about custom exceptions?

Answer

Custom exceptions must extend either Exception or RuntimeException

Status : Correct **Marks : 1/1**

6. What will be the output for the following code?

```
import java.io.*;

class UnderageException extends Exception {
   public UnderageException(String message) {
      super(message);
   }
}

class Test {
   public static void main(String[] args) {
      try {
         int age = 17;
         if (age < 18) {
            throw new UnderageException("Underage, cannot proceed");
         }
      } catch (UnderageException e) {
         System.out.println(e.getMessage());
      }
   }
}
```

Answer

Underage, cannot proceed

Status : Correct ***Marks : 1/1***

7. What is the purpose of a custom exception in Java?

Answer

To create user-defined exceptions for specific scenarios

Status : Correct ***Marks : 1/1***

8. How do you create an unchecked custom exception?

Answer

By extending RuntimeException

Status : Correct **Marks : 1/1**

9. What will be the output for the following code?

```
class NegativeBalanceException extends Exception {
   public NegativeBalanceException(String message) {
      super(message);
   }
}

class Test {
   public static void main(String[] args) {
      try {
         double balance = -500;
         if (balance < 0) {
            throw new NegativeBalanceException("Balance cannot be
negative");
         }
      } catch (NegativeBalanceException e) {
         System.out.println("Error: " + e.getMessage());
      }
   }
}
```

Answer

Error: Balance cannot be negative

Status : Correct **Marks : 1/1**

10. What will be the output of the following code?

```
class MyException extends Exception {
   public MyException() {
      super("Default Exception Message");
   }
}

class Test {
```

```
    public static void main(String[] args) {
        try {
            throw new MyException();
        } catch (MyException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Answer

Default Exception Message

Status : Correct ***Marks : 1/1***

11. Which keyword is used to explicitly throw a custom exception?

Answer

throw

Status : Correct ***Marks : 1/1***

12. what is the output of the following code?

```
class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}

class Test {
    static void check() throws MyException {
        throw new MyException("Custom Exception Occurred");
    }

    public static void main(String[] args) {
        try {
            check();
        } catch (Exception e) {
```

```
            System.out.println(e.getMessage());
        }
    }
}
```

Answer

Custom Exception Occurred

Status : Correct ***Marks : 1/1***

13. What will be the output for the following code?

```
import java.io.*;

class TemperatureTooHighException extends Exception {
    public TemperatureTooHighException(String message) {
        super(message);
    }
}

class Test {
    public static void main(String[] args) {
        try {
            int temperature = 110;
            if (temperature > 100) {
                throw new TemperatureTooHighException("Temperature too
high");
            }
        } catch (TemperatureTooHighException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Answer

Temperature too high

Status : Correct ***Marks : 1/1***

14. What will be the output for the following code?

```
import java.io.*;

class OutOfStockException extends Exception {
   public OutOfStockException(String message) {
      super(message);
   }
}

class Test {
   public static void main(String[] args) {
      try {
         int stock = 0;
         if (stock == 0) {
            throw new OutOfStockException("Item is out of stock");
         }
      } catch (OutOfStockException e) {
         System.out.println(e.getMessage());
      }
   }
}
```

Answer

Item is out of stock

Status : Correct ***Marks : 1/1***

15. What will be the output for the following code?

```
class InvalidVotingAgeException extends Exception {
   public InvalidVotingAgeException(String message) {
      super(message);
   }
}

class Test {
   public static void main(String[] args) {
      try {
```

```
            int age = 15;
            if (age < 18) {
                throw new InvalidVotingAgeException("You are not eligible to
vote");
            }
            System.out.println("Eligible to vote");
        } catch (InvalidVotingAgeException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Answer

You are not eligible to vote

Status : Correct ***Marks : 1/1***

Rajalakshmi Engineering College

Name: NISHANTH ELANGO RAJAN
Email: 241901075@rajalakshmi.edu.in
Roll no: 241901075
Phone: 9444909050
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 8_PAH

Attempt : 2
Total Mark : 40
Marks Obtained : 40

Section 1 : Coding

1. Problem Statement

You are tasked to create a program that defines a custom exception GradeException. The program should include a Student class with fields for the student's name, age, and grade. Implement a method in the Student class that checks the grade, and if the grade is below 40, it should throw a GradeException. Otherwise, it should display the student's details.

Input Format

The input consists of three parameters in separate lines:

1. A string representing the student's name.
2. An integer representing the student's age.
3. An integer representing the student's grade.

Output Format

The output will display the student's details if the grade is valid.

If the grade is below 40, the program will display an error message "Grade is below 40".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: Alice
20
85
Output: Name: Alice
Age: 20
Grade: 85

Answer

```
import java.util.Scanner;
class GradeException extends Exception {
    GradeException(String message) {
        System.out.println(message);
    }
}
class Student {
    String name;
    int age;
    int grade;
    Student(String studentName, int studentAge, int studentGrade) {
        name = studentName;
        age = studentAge;
        grade = studentGrade;
    }
    void validateGrade() throws GradeException {
        if (grade < 40) {
            throw new GradeException("Grade is below 40");
        }
    }
    void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
```

```
        System.out.println("Grade: " + grade);
    }
}
public class Main {
    public static void main(String[] args) {
        try {
            java.util.Scanner scanner = new java.util.Scanner(System.in);
            String studentName = scanner.nextLine();
            int studentAge = scanner.nextInt();
            int studentGrade = scanner.nextInt();
            Student student = new Student(studentName, studentAge, studentGrade);
            student.validateGrade();
            student.displayDetails();
            scanner.close();
        } catch (GradeException e) {

        }
    }
}
```

Status : Correct **Marks : 10/10**

2. Problem Statement

An HR software system is being developed to process employee payrolls. During payroll processing, the system must ensure that no employee has a negative salary and that no employee's salary exceeds 2,00,000. If either condition occurs, the system should throw a custom exception.

Create a custom exception InvalidSalaryException and a class Employee that processes salary according to the following rules:

If salary < 0, throw InvalidSalaryException with the message: "Salary cannot be negative". If salary > 200000, throw InvalidSalaryException with the message: "Salary exceeds threshold limit". Otherwise, display: "Salary processed successfully for <empName>: <salary>".

The payroll processing should always display: "Payroll process completed" at the end, regardless of whether an exception occurs.

Input Format

The first line of input contains an integer representing the employee ID.

The second line contains a string representing the employee's name.

The third line contains a floating-point number representing the salary of the employee.

Output Format

If the salary is valid: "Salary processed successfully for <empName>: <salary>"

"Payroll process completed"

If the salary is invalid: "<Exception Message>"

"Payroll process completed"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 101
Rahul
150000.0
Output: Salary processed successfully for Rahul: 150000.0
Payroll process completed

Answer

```
import java.util.Scanner;
class InvalidSalaryException extends Exception {
    public InvalidSalaryException(String message) {
        super(message);
    }
}
class Employee {
    int empId;
    String empName;
    double salary;
    public Employee(int empId, String empName, double salary) {
```

```
        this.empId = empId;
        this.empName = empName;
        this.salary = salary;
    }
    public void processSalary() throws InvalidSalaryException {
        if (salary < 0) {
            throw new InvalidSalaryException("Salary cannot be negative");
        }
        if (salary > 200000) {
            throw new InvalidSalaryException("Salary exceeds threshold limit");
        }
        System.out.println("Salary processed successfully for " + empName + ": " +
salary);
    }
}
class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            int empId = scanner.nextInt();
            scanner.nextLine();
            String empName = scanner.nextLine();
            double salary = scanner.nextDouble();
            Employee employee = new Employee(empId, empName, salary);
            try {
                employee.processSalary();
            } catch (InvalidSalaryException e) {
                System.out.println(e.getMessage());
            }
        } finally {
            System.out.println("Payroll process completed");
            scanner.close();
        }
    }
}
```

Status : Correct **Marks : 10/10**

3. Problem Statement

Daniel is developing a program to verify the age of users. He wants to

ensure that the entered age is within a valid range. Write a program to help Daniel implement this age-checking feature using custom exceptions.

Daniel needs a program that takes an integer input representing a person's age.If the age is between 0 and 150 (inclusive), the program should print "Age is valid!".If the age is less than 0 or greater than 150, the program should throw a custom exception (InvalidAgeException) with the message "Invalid age. Please enter an age between 0 and 150."

Implement a custom exception, InvalidAgeException, to handle cases where the entered age does not meet the specified criteria.

Input Format

The input consists of an integer value 'n', representing the age.

Output Format

The output is displayed in the following format:

If the age is valid (between 0 and 150, inclusive), print

"Age is valid!".

If the age is invalid, print

"Error: Invalid age. Please enter an age between 0 and 150."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 45
Output: Age is valid!

Answer

```
import java.util.Scanner;
class AgeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
```

```
            int age = scanner.nextInt();
            checkAge(age);
            System.out.println("Age is valid!");
        } catch (InvalidAgeException | java.util.InputMismatchException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
    private static void checkAge(int age) throws InvalidAgeException {
        if (age < 0 || age > 150) {
            throw new InvalidAgeException("Invalid age. Please enter an age between
0 and 150.");
        }
    }
}
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}
```

Status : Correct **Marks : 10/10**

4. Problem Statement

Enigma is developing a simple web application that takes a user-input URL, validates it, and throws a custom exception InvalidURLFormatException if the URL does not start with "http://" or "https://".

The main method prompts the user for input, validates the URL, and prints whether it is valid or not.

Input Format

The input consists of a string, representing the URL entered by the user.

Output Format

The output displays one of the following results:

If the entered URL is valid according to the specified format, the program prints:

"[URL] is a valid URL"

If the entered URL is not valid according to the specified format, the program prints:

"Invalid URL format: [URL]"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: http://www.example.com

Output: http://www.example.com is a valid URL

Answer

```
import java.util.Scanner;
class WebApplication {
    public static void main(String[] args) {
        String userInputURL = getUserInputURL();
        try {
            validateURL(userInputURL);
            System.out.println(userInputURL+" is a valid URL");
        } catch (InvalidURLFormatException e) {
            System.out.println(e.getMessage()+userInputURL);
        }
    }
    private static String getUserInputURL() {
        Scanner scanner = new Scanner(System.in);
        return scanner.nextLine();
    }
    private static void validateURL(String url) throws InvalidURLFormatException {
        if (!(url.startsWith("http://") || url.startsWith("https://"))) {
            throw new InvalidURLFormatException("Invalid URL format: ");
        }
    }
}
```

```
class InvalidURLFormatException extends Exception {
    public InvalidURLFormatException(String message) {
        super(message);
    }
}
```

Status : Correct **Marks : 10/10**