Abdul Hasib Zahid
Id: 443058333

Experiment # 18: Custom Exception - *DivisionByZeroException*

You're working on a simple calculator that needs to handle division. Create a `DivisionByZeroException` custom exception to handle the case where division by zero is attempted.

Instructions:

- **Create a `DivisionByZeroException` class:** This class must inherit from `ArithmeticException`.
- Add a constructor that takes a message as parameter.
-
- **Create a `divide(int numerator, int denominator)` method:** Create a `divide(int numerator, int denominator)` method that divides two numbers.
- If the denominator is zero, throw the `DivisionByZeroException` with an appropriate message.
-
- **Test your exception:** Test the exception in the main method.

```
class DivisionByZeroException extends ArithmeticException {
    public DivisionByZeroException(String message) {
        super(message);
    }
}

public class Calculator {
    public static double divide(int numerator, int denominator) throws DivisionByZeroException {
        if (denominator == 0) {
            throw new DivisionByZeroException("Cannot divide by zero.");
        }
        return (double) numerator / denominator;
    }

    public static void main(String[] args) {
        try {
            System.out.println("Result: " + divide(10, 0));
        } catch (DivisionByZeroException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Experiment # 19: Custom Exception - Input Validation

Develop a form management application with validation of data entered by the user using a custom exception `InvalidInputException`.

Instructions:

- **Create the `InvalidInputException` class:** Define an `InvalidInputException` class that inherits from `Exception`.
- Add a constructor that takes a message as a parameter to specify the reason of the exception.
-
- **Creating the validateInput method:** Create a `validateInput(String input)` method that checks whether the input is empty or null.
- If the input is empty or null, throw an `InvalidInputException` with an appropriate message.
-
- **Writing the main program :** Write a main program that asks the user to enter text.
- Call the `validateInput` method with the user's input.
- Catch the `InvalidInputException` and display the error message if an exception is thrown.

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

public class FormManager {
   public static void validateInput(String input) throws InvalidInputException{
      if(input == null || input.isEmpty()){
         throw new InvalidInputException("Input cannot be empty or null.");
      }
   }

   public static void main(String[] args) {
      java.util.Scanner scanner = new java.util.Scanner(System.in);
      System.out.println("Enter text: ");
      String userInput = scanner.nextLine();

      try{
         validateInput(userInput);
         System.out.println("Valid Input: " + userInput);
      } catch(InvalidInputException e){
         System.out.println("Error: "+e.getMessage());
      }
   }
}
```

Experiment # 20: Custom Exception - Inventory Management

Develop an inventory management system for a store using a custom exception `OutOfStockException` to handle insufficient stock errors on orders.

Instructions:

- **Creating the `OutOfStockException` class:** Define an `OutOfStockException` class that inherits from `Exception`.
- Add a constructor that takes a message as a parameter to specify the reason of the exception.
-
- **Creating the checkStock method:** Create a `checkStock(int available, int requested)` method that checks whether the requested quantity is available.
- If the requested quantity exceeds the available stock, throw the `OutOfStockException` with an appropriate message.
-
- **Writing the main program :** Write a main program that simulates a situation where the user tries to order more products than the available stock.
- Call the `checkStock` method with the quantities available and requested.
- Catch the `OutOfStockException` and display the error message if an exception is thrown.

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

public class InventoryManager {
    public static void checkStock(int available, int requested) throws OutOfStockException{
        if(requested > available){
            throw new OutOfStockException("Insufficient stock available. Requested: " + requested + ",
Available: " + available);
        }
    }

    public static void main(String[] args) {
        int availableStock = 10;
        int requestedQuantity = 15;

        try{
            checkStock(availableStock, requestedQuantity);
        } catch (OutOfStockException e){
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```