Lab 2 - Java

ESS 201 Programming II

International Institute of Information Technology – Bangalore

Part A (Submission: LMS by 15 Sep 11:00:00)

To be worked on in the Lab session. The source code for this exercise should be submitted on LMS.

Task 1:

Create a Java program for an e-commerce platform that manages products. Each product needs a unique ID, and this ID should be generated automatically when a new product is created. You will implement this functionality using constructors and constructor overloading.

Requirements:

1. Create a class named "Product" with the following attributes:
    - int productId (unique identifier for each product)
    - String productName
    - double price
2. Implement multiple constructors for the "Product" class as follows:

    a) A parameterless constructor that initializes the product with default values.

    b) A constructor that takes the product name and price as parameters, and assigns them to the respective attributes. The productId should be generated incrementally, starting from 1 for the first product and incrementing by 1 for each new product.

    c) A constructor that takes only one parameters: productName.
3. Implement a method within the "Product" class to display the product details, including productId, productName, and price.
4. Create a "Main" class with the "main" method to demonstrate the functionality of the "Product" class.
5. In the "Main" class, create instances of the "Product" class using different constructors. Display the product details for each product to verify that the constructors and the incremental productId generation are working correctly.

Sample Input:

3

Laptop

799.99

Smartphone

Headphones

200

Sample Output

Product Details:

Product ID: 1

Product Name: Laptop

Price: $799.99

Product Details:

Product ID: 2

Product Name: Smartphone

Price: 0

Product Details:

Product ID: 3

Product Name: Headphones

Price: 200

Note:

- Use below statement to code to generate unique product IDs:

```
private static int nextProductId = 1;
```

- Use below code to take input in java:

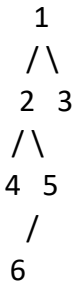
```
import java.util.Scanner;

Scanner scanner = new Scanner(System.in);
scanner.nextLine();
scanner.nextDouble();
scanner.nextInt();
```

Task 2:

A Tree data structure has a simple recursive structure. We can model this as a Java class, where each node has references to instances of the same Tree class. In this assignment, you are required to create a binary tree using a Java class called **Tree**. Each **Tree** node should have two children, **leftChild** and **rightChild**, both also of type **Tree**. The **Tree** class should also contain an integer value, **data**. Note that either or both of the children can be null.

Tasks:

1. Implement a **Tree** class with the following specifications:
   - The **Tree** class should have a constructor that takes two **Tree** objects as arguments - the left and right child. Either or both of these children could be null.
   - Implement a method called **preOrder** that will traverse the tree (recursively) and print out the values in pre-order.
2. In your **main** method, create each instance of the **Tree** nodes and construct the tree structure by setting the appropriate children for each node. Your task is to create the following tree structure:



Sample Output:
1 2 4 5 6 3

Part B (Submission: LMS by 29 Sep 11:00:00)

The source code for this exercise should be submitted by the due date.

Extend the previously defined **Tree** class to include additional methods for different tree traversal techniques and tree-related operations. You should continue to use the binary tree structure with left and right children, and each node contains an integer value.

Tasks:

1. Implement the following methods in the **Tree** class:
    - **preOrder**: Traverses the tree in pre-order and prints the values.
    - **inOrder**: Traverses the tree in in-order and prints the values.
    - **postOrder**: Traverses the tree in post-order and prints the values.
    - **levelOrder**: Traverses the tree in level order (breadth-first) and prints the values.
    - **getSize**: Returns the total number of nodes in the tree.
    - **getHeight**: Returns the height of the tree (the length of the longest path from the root to a leaf node).
    - **contains(int value)**: Checks if the tree contains a node with the given value and returns **true** if found, **false** otherwise.
    - **findMax**: Finds and returns the maximum value in the tree.
    - **findMin**: Finds and returns the minimum value in the tree.
    - **insert(int value)**: Inserts a new node with the given value into the tree.
2. In your **main** method, create an instance of the **Tree** class and construct the tree structure by setting the appropriate children for each node. You can choose any values and structure for your tree.