**ASSIGNMENT 1:** BUILD A SIMPLE TEXT-BASED GAME

Objective: Reinforce object-oriented programming concepts. Implement game logic, user input, and output. Explore basic game development principles.

Requirements: Create classes for player, enemies, items, and game world. Implement game mechanics like movement, combat, and inventory management. Use basic input/output for user interaction. Consider adding features like levels, scorekeeping, and difficulty levels.

**Solution:**

**Player.java**

**package** pack1;

**public** **class** Player {

**private** String name;

**private** **int** health;

**private** **int** x, y; // Player coordinates

**private** **int** inventoryCapacity;

**private** Item[] inventory;

**public** Player(String name, **int** health, **int** x, **int** Y, **int** inventoryCapacity) {

**this**.name = name;

**this**.health = health;

**this**.x = x;

**this**.y = y;

**this**.inventoryCapacity = inventoryCapacity;

**this**.inventory = **new** Item[inventoryCapacity];

}

**public** **void** move(String direction) {

// Implement movement logic

**if** (direction.equals("up")) {

**this**.y = Math.*max*(0, **this**.y - 1);

} **else** **if** (direction.equals("down")) {

**this**.y = Math.*min*(Game.***WIDTH*** - 1, **this**.y + 1);

} **else** **if** (direction.equals("left")) {

**this**.x = Math.*max*(0, **this**.x - 1);

} **else** **if** (direction.equals("right")) {

**this**.x = Math.*min*(Game.***HEIGHT*** - 1, **this**.x + 1);

}

}

**public** **void** attack(Enemy enemy) {

// Implement attack logic

**int** damage = (**int**) (Math.*random*() \* 10) + 10; // Random damage between 10 to 19

System.***out***.println("You attack the " + enemy.getName() + " and deal " + damage + " damage!");

enemy.takeDamage(damage);

}

**public** **void** collectItem(Item item) {

// Implement item collection logic

**for** (**int** i = 0; i < inventoryCapacity; i++) {

**if** (inventory[i] == **null**) {

inventory[i] = item;

System.***out***.println("You collected a " + item.getName());

**break**;

}

}

}

**public** **boolean** isAlive() {

**return** health > 0;

}

**public** **void** takeDamage(**int** damage) {

// **TODO** Auto-generated method stub

}

**public** **int** getX() {

// **TODO** Auto-generated method stub

**return** 0;

}

**public** **int** getY() {

// **TODO** Auto-generated method stub

**return** 0;

}

// Getters and setters

}

**Enemy.java:**

**public** **class** Enemy {

**private** String name;

**private** **int** health;

**public** Enemy(String name, **int** health) {

**this**.name = name;

**this**.health = health;

}

**public** **void** takeDamage(**int** damage) {

**this**.health -= damage;

**if**(**this**.health <= 0) {

System.***out***.println("The " + **this**.name + " has been defeated!");

}

}

**public** **void** attack(Player player) {

// Implement enemy attack logic

**int** damage = (**int**) (Math.*random*() \* 5) + 5; // Random damage between 5 to 9

System.***out***.println("The " + **this**.name + " attacks you and deals " + damage + " damage!");

player.takeDamage(damage);

}

**public** String getName() {

// **TODO** Auto-generated method stub

**return** **null**;

}

// Getters and setters

}

**Item.java:**

**public** **class** Item {

**private** String name;

**public** Item(String name) {

**this**.name = name;

}

**public** String getName() {

**return** name;

}

// Getters and setters

}

**Room.java:**

**public** **class** Room {

**private** Enemy enemy;

**private** Item item;

**private** **boolean** visited;

**public** Room(Enemy enemy, Item item) {

**this**.enemy = enemy;

**this**.item = item;

**this**.visited = **false**;

}

**public** **void** enter(Player player) {

**if** (!visited) {

System.***out***.println("You enter the room.");

**if** (enemy != **null**) {

enemy.attack(player);

}

**if** (item != **null**) {

player.collectItem(item);

}

visited = **true**;

} **else** {

System.***out***.println("You've already visited this room.");

}

}

// Getters and setters

}

**Game.java:**

**import** java.util.Scanner;

**public** **class** Game {

**public** **static** **final** **int** ***WIDTH*** = 5;

**public** **static** **final** **int** ***HEIGHT*** = 5;

**private** Player player;

**private** Room[][] dungeon;

**private** **boolean** gameOver;

**public** Game() {

**this**.player = **new** Player("Player", 100, 0, 0, 5); // Start at position (0, 0)

**this**.dungeon = **new** Room[***WIDTH***][***HEIGHT***];

**this**.gameOver = **false**;

// Initialize rooms with enemies and items

// For simplicity, randomly place enemies and items in rooms

**for**(**int** i = 0; i < ***WIDTH***; i++) {

**for**(**int** j = 0; j < ***HEIGHT***; j++) {

Enemy enemy = **null**;

Item item = **null**;

// 30% chance of an enemy

**if**(Math.*random*() < 0.3) {

enemy = **new** Enemy("Monster", (**int**) (Math.*random*() \* 50) + 50); // Random health between 50 to 99

}

// 20% chance of an item

**if**(Math.*random*() < 0.2) {

item = **new** Item("Potion");

}

dungeon[i][j] = **new** Room(enemy, item);

}

}

}

**public** **void** start() {

Scanner scanner = **new** Scanner(System.***in***);

**while**(!gameOver) {

System.***out***.println("Enter your move (up/down/left/right): ");

String move = scanner.nextLine();

// Process player's move

player.move(move);

// Get current room

Room currentRoom = dungeon[player.getX()][player.getY()];

// Enter the current room

currentRoom.enter(player);

// Check if game over condition (player health <= 0)

**if**(!player.isAlive()) {

System.***out***.println("Game Over! You have been defeated.");

gameOver = **true**;

}

// Check if player reached the exit (bottom-right corner)

**if**(player.getX() == ***WIDTH*** - 1 && player.getY() == ***HEIGHT*** - 1) {

System.***out***.println("Congratulations! You have reached the exit.");

gameOver = **true**;

}

}

scanner.close();

}

**public** **static** **void** main(String[] args) {

Game game = **new** Game();

game.start();

}

}

**ASSIGNMENT 2:** CREATE A FILE I/O APPLICATION

Objective: Understand file handling in Java. Practice reading and writing data to files. Learn about different file formats (e.g., CSV, JSON, XML).

Requirements: Create a program to read data from a CSV file and store it in objects. Implement functionality to write data from objects to a JSON file. Handle exceptions for file-related operations. Explore using file filters and directory traversal

**Solution:**

**Xml:**

**<dependencies>**

**<dependency>**

**<groupId>org.apache.commons</groupId>**

**<artifactId>commons-csv</artifactId>**

**<version>1.9</version>**

**</dependency>**

**<dependency>**

**<groupId>com.googlecode.json-simple</groupId>**

**<artifactId>json-simple</artifactId>**

**<version>1.1.1</version>**

**</dependency>**

**</dependencies>**

**File Handling Code:**

**package** pack1;

**import** org.apache.commons.csv.CSVFormat;

**import** org.apache.commons.csv.CSVParser;

**import** org.apache.commons.csv.CSVRecord;

**import** org.json.simple.JSONArray;

**import** org.json.simple.JSONObject;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.io.Reader;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** FileIOExample {

**public** **static** **void** main(String[] args) {

String csvFile = "input.csv";

String jsonFile = "output.json";

// Read data from CSV and store in objects

List<Employee> employees = *readCSV*(csvFile);

// Write data from objects to JSON

*writeJSON*(jsonFile, employees);

System.***out***.println("File operations completed successfully.");

}

// Employee class for storing data from CSV

**static** **class** Employee {

**private** String name;

**private** **int** age;

**private** String department;

**public** Employee(String name, **int** age, String department) {

**this**.name = name;

**this**.age = age;

**this**.department = department;

}

// Getters and setters (not shown for brevity)

}

// Method to read data from CSV into a list of Employee objects

**private** **static** List<Employee> readCSV(String csvFile) {

List<Employee> employees = **new** ArrayList<>();

**try**(Reader reader = **new** FileReader(csvFile);

CSVParser csvParser = **new** CSVParser(reader, CSVFormat.DEFAULT)) {

**for**(CSVRecord csvRecord : csvParser) {

String name = csvRecord.get(0);

**int** age = Integer.*parseInt*(csvRecord.get(1));

String department = csvRecord.get(2);

employees.add(**new** Employee(name, age, department));

}

} **catch** (IOException e) {

e.printStackTrace();

}

**return** employees;

}

// Method to write data from a list of Employee objects to a JSON file

**private** **static** **void** writeJSON(String jsonFile, List<Employee> employees) {

JSONArray employeeList = **new** JSONArray();

**for** (Employee emp : employees) {

JSONObject employee = **new** JSONObject();

employee.put("name", emp.name);

employee.put("age", emp.age);

employee.put("department", emp.department);

employeeList.add(employee);

}

**try** (FileWriter fileWriter = **new** FileWriter(jsonFile)) {

fileWriter.write(employeeList.toJSONString());

fileWriter.flush();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**ASSIGNMENT 3**: IMPLEMENT A SIMPLE WEB APPLICATION

Objective: Introduce to web development using Java Servlets and JSP. Understand the basics of HTTP and web application architecture. Learn to handle requests and responses.

Requirements: Create a web application that displays dynamic content. Use Servlets to handle requests and generate responses. Employ JSP for creating HTML pages with embedded Java code. Explore session management and request attributes

SOLUTION:

Servlet:

**package** Example;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.annotation.WebServlet;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

/\*\*

\* Servlet implementation class HelloServlet

\*/

**public** **class** HelloServlet **extends** HttpServlet {

@Override

**protected** **void** doGet(Example.HttpServlet request, Example.HttpServlet response)

**throws** IOException {

// Set response content type

response.setContentType("text/html");

// Get or create session

HttpSession session = request.getSession();

String userName = (String) session.getAttribute("userName");

// If no session attribute, create a default

**if** (userName == **null**) {

userName = "Guest";

}

// Set the user name as a request attribute

request.setAttribute("userName", userName);

// Forward to JSP page

((Object) request.getRequestDispatcher("hello.jsp")).forward(request, response);

}

}

**SecondServelet.java:**

Package Example;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@WebServlet("/setUserName")

public class SetUserNameServlet extends HttpServlet {

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Get the user name from the form

String userName = request.getParameter("name");

// Get or create session

HttpSession session = request.getSession();

session.setAttribute("userName", userName);

// Redirect to the HelloServlet

response.sendRedirect("hello");

}

}

**JSP.JAVA:**

**<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Hello JSP</title>**

**</head>**

**<body>**

**<h1>Hello, <%= request.getAttribute("userName") %>!</h1>**

**<form action="setUserName" method="post">**

**<label for="name">Enter your name:</label>**

**<input type="text" id="name" name="name" required>**

**<input type="submit" value="Submit">**

**</form>**

**</body>**

**</html>**

**Web.xml:**

**<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee**

**http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"**

**version="3.1">**

**<servlet>**

**<servlet-name>HelloServlet</servlet-name>**

**<servlet-class>com.example.HelloServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>HelloServlet</servlet-name>**

**<url-pattern>/hello</url-pattern>**

**</servlet-mapping>**

**<servlet>**

**<servlet-name>SetUserNameServlet</servlet-name>**

**<servlet-class>com.example.SetUserNameServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>SetUserNameServlet</servlet-name>**

**<url-pattern>/setUserName</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**ASSIGNMENT 4:** IMPLEMENT A BINARY SEARCH TREE (BST)

Implement a Binary Search Tree (BST) with the following operations:

* Insert a node
* Delete a node
* Search for a node
* In-order traversal
* Pre-order traversal
* Post-order traversal
* Ensure the tree maintains the BST properties after each operation.

**SOLUTION:**

**NODE CLASS:**

**package** pack1;

**public** **class** Node {

**int** value;

Node left, right;

**public** Node(**int** value) {

**this**.value = value;

**this**.left = **this**.right = **null**;

}

}

**BinarySearchTree.java:**

**public** **class** BinarySearchTree {

**private** Node root;

// Constructor

**public** BinarySearchTree() {

**this**.root = **null**;

}

// Insert a node

**public** **void** insert(**int** value) {

root = insertRec(root, value);

}

**private** Node insertRec(Node node, **int** value) {

**if** (node == **null**) {

node = **new** Node(value);

**return** node;

}

**if** (value < node.value) {

node.left = insertRec(node.left, value);

} **else** **if** (value > node.value) {

node.right = insertRec(node.right, value);

}

**return** node;

}

// Delete a node

**public** **void** delete(**int** value) {

root = deleteRec(root, value);

}

**private** Node deleteRec(Node root, **int** value) {

**if** (root == **null**) {

**return** root;

}

**if** (value < root.value) {

root.left = deleteRec(root.left, value);

} **else** **if** (value > root.value) {

root.right = deleteRec(root.right, value);

} **else** {

**if** (root.left == **null**) {

**return** root.right;

} **else** **if** (root.right == **null**) {

**return** root.left;

}

root.value = minValue(root.right);

root.right = deleteRec(root.right, root.value);

}

**return** root;

}

**private** **int** minValue(Node root) {

**int** minValue = root.value;

**while** (root.left != **null**) {

minValue = root.left.value;

root = root.left;

}

**return** minValue;

}

// Search for a node

**public** **boolean** search(**int** value) {

**return** searchRec(root, value);

}

**private** **boolean** searchRec(Node node, **int** value) {

**if** (node == **null**) {

**return** **false**;

}

**if**(node.value == value) {

**return** **true**;

}

**return** value < node.value ? searchRec(node.left, value):searchRec(node.right, value);

}

// In-order traversal

**public** **void** inOrderTraversal() {

inOrderTraversalRec(root);

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

}

**private** **void** inOrderTraversalRec(Node node) {

**if** (node != **null**) {

inOrderTraversalRec(node.left);

System.***out***.print(node.value + " ");

inOrderTraversalRec(node.right);

}

}

// Pre-order traversal

**public** **void** preOrderTraversal() {

preOrderTraversalRec(root);

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

}

**private** **void** preOrderTraversalRec(Node node) {

**if** (node != **null**) {

System.***out***.print(node.value + " ");

preOrderTraversalRec(node.left);

PreOrderTraversalRec(node.right);

}

}

// Post-order traversal

**public** **void** postOrderTraversal() {

postOrderTraversalRec(root);

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

}

**private** **void** postOrderTraversalRec(Node node) {

**if** (node != **null**) {

postOrderTraversalRec(node.left);

postOrderTraversalRec(node.right);

System.***out***.print(node.value + " ");

}

}

// Main method for testing

**public** **static** **void** main(String[] args) {

BinarySearchTree bst = **new** BinarySearchTree();

bst.insert(50);

bst.insert(30);

bst.insert(20);

bst.insert(40);

bst.insert(70);

bst.insert(60);

bst.insert(80);

System.***out***.println("In-order traversal:");

bst.inOrderTraversal(); // Output should be: 20 30 40 50 60 70 80

System.***out***.println("Pre-order traversal:");

bst.preOrderTraversal(); // Output should be: 50 30 20 40 70 60 80

System.***out***.println("Post-order traversal:");

bst.postOrderTraversal(); // Output should be: 20 40 30 60 80 70 50

System.***out***.println("Search for 40: " + bst.search(40)); // Output should be: true

System.***out***.println("Search for 90: " + bst.search(90)); // Output should be: false

bst.delete(20);

System.***out***.println("In-order traversal after deleting 20:");

bst.inOrderTraversal(); // Output should be: 30 40 50 60 70 80

bst.delete(30);

System.***out***.println("In-order traversal after deleting 30:");

bst.inOrderTraversal(); // Output should be: 40 50 60 70 80

}

}

**ASSIGNMENT 5:** STACK IMPLEMENTATION USING ARRAYS

Requirements:

* Implement a stack using an array with the following operations:
* Push an element onto the stack.
* Pop an element from the stack.
* Peek at the top element of the stack.
* Check if the stack is empty.
* Check if the stack is full

**SOLUTION:**

**package** pack1;

**public** **class** Stack {

**private** **int**[] stack;

**private** **int** top;

**private** **int** capacity;

// Constructor to initialize the stack with a specific capacity

**public** Stack(**int** capacity) {

**this**.capacity = capacity;

**this**.stack = **new** **int**[capacity];

**this**.top = -1; // Indicates an empty stack

}

// Push an element onto the stack

**public** **void** push(**int** value) {

**if** (isFull()) {

System.***out***.println("Stack is full. Cannot push " + value);

**return**;

}

stack[++top] = value;

System.***out***.println("Pushed " + value);

}

// Pop an element from the stack

**public** **int** pop() {

**if** (isEmpty()) {

System.***out***.println("Stack is empty. Cannot pop.");

**return** -1; // Returning -1 to indicate the stack is empty

}

**return** stack[top--];

}

// Peek at the top element of the stack

**public** **int** peek() {

**if** (isEmpty()) {

System.***out***.println("Stack is empty. Nothing to peek.");

**return** -1; // Returning -1 to indicate the stack is empty

}

**return** stack[top];

}

// Check if the stack is empty

**public** **boolean** isEmpty() {

**return** top == -1;

}

// Check if the stack is full

**public** **boolean** isFull() {

**return** top == capacity - 1;

}

// Main method to test the stack implementation

**public** **static** **void** main(String[] args) {

Stack stack = **new** Stack(5); // Create a stack with capacity of 5

stack.push(10);

stack.push(20);

stack.push(30);

System.***out***.println("Top element is: " + stack.peek());

System.***out***.println("Popped element: " + stack.pop());

System.***out***.println("Popped element: " + stack.pop());

System.***out***.println("Is stack empty? " + stack.isEmpty());

stack.push(40);

stack.push(50);

stack.push(60);

stack.push(70); // This will fail as the stack is full

System.***out***.println("Is stack full? " + stack.isFull());

System.***out***.println("Popped element: " + stack.pop());

System.***out***.println("Popped element: " + stack.pop());

System.***out***.println("Popped element: " + stack.pop());

System.***out***.println("Popped element: " + stack.pop()); // This will indicate the stack is empty

}

}