**WEEK 4 – LAB PROBLEM**

Q1. Virtual Pet Evolution Simulator

Soln:

import java.util.\*;

public class VirtualPet {

private final String petId;

private String petName;

private String species;

private int age;

private int happiness;

private int health;

private int stageIndex;

static final String[] EVOLUTION\_STAGES = {"Egg", "Baby", "Child", "Teen", "Adult", "Elder"};

static int totalPetsCreated = 0;

public VirtualPet() {

this("Unknown", getRandomSpecies(), 0, 50, 50, 0);

}

public VirtualPet(String petName) {

this(petName, getRandomSpecies(), 0, 70, 70, 1);

}

public VirtualPet(String petName, String species) {

this(petName, species, 2, 80, 80, 2);

}

public VirtualPet(String petName, String species, int age, int happiness, int health, int stageIndex) {

this.petId = generatePetId();

this.petName = petName;

this.species = species;

this.age = age;

this.happiness = happiness;

this.health = health;

this.stageIndex = stageIndex;

totalPetsCreated++;

}

public void evolvePet() {

if (species.equals("Ghost")) return;

if (stageIndex < EVOLUTION\_STAGES.length - 1 && age > stageIndex \* 2 && happiness > 30 && health > 30) {

stageIndex++;

System.out.println(petName + " has evolved into " + EVOLUTION\_STAGES[stageIndex] + " stage!");

}

}

public void feedPet() {

if (species.equals("Ghost")) return;

happiness += 5;

health += 10;

System.out.println(petName + " was fed. Happiness: " + happiness + " Health: " + health);

}

public void playWithPet() {

if (species.equals("Ghost")) return;

happiness += 10;

health -= 5;

System.out.println(petName + " played. Happiness: " + happiness + " Health: " + health);

}

public void healPet() {

if (species.equals("Ghost")) return;

health += 20;

System.out.println(petName + " was healed. Health: " + health);

}

public void simulateDay() {

if (species.equals("Ghost")) return;

age++;

happiness -= (int)(Math.random() \* 5);

health -= (int)(Math.random() \* 5);

if (health <= 0) {

species = "Ghost";

stageIndex = -1;

System.out.println(petName + " has died and become a Ghost!");

} else {

evolvePet();

}

}

public String getPetStatus() {

if (species.equals("Ghost")) return petName + " [Ghost]";

return petName + " (" + species + ") - Age: " + age + " Stage: " + EVOLUTION\_STAGES[stageIndex] +

" Happiness: " + happiness + " Health: " + health;

}

public static String generatePetId() {

return UUID.randomUUID().toString();

}

private static String getRandomSpecies() {

String[] speciesOptions = {"Dragon", "Phoenix", "Wolf", "Cat", "Dog"};

return speciesOptions[new Random().nextInt(speciesOptions.length)];

}

public static void main(String[] args) {

List<VirtualPet> pets = new ArrayList<>();

pets.add(new VirtualPet());

pets.add(new VirtualPet("Buddy"));

pets.add(new VirtualPet("Luna", "Wolf"));

pets.add(new VirtualPet("Max", "Dragon", 5, 90, 90, 3));

for (int day = 1; day <= 5; day++) {

System.out.println("\n--- Day " + day + " ---");

for (VirtualPet pet : pets) {

pet.simulateDay();

System.out.println(pet.getPetStatus());

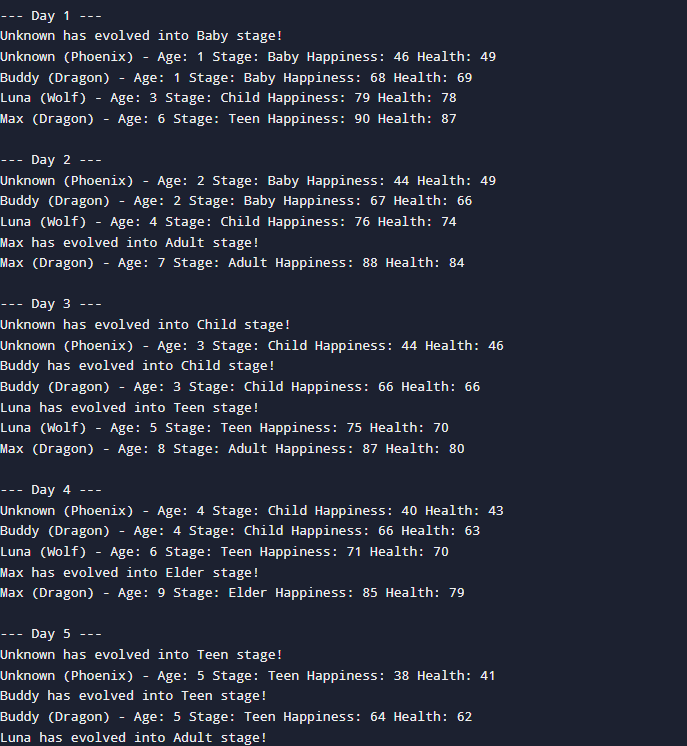
}

}

System.out.println("\nTotal pets created: " + VirtualPet.totalPetsCreated);

}

}

SOLN. 

Q2. Medieval Kingdom Builder with Magic System

Soln.

import java.util.\*;

public class MagicalKingdom {

abstract static class MagicalStructure {

protected String structureName;

protected int magicPower;

protected String location;

protected boolean isActive;

public MagicalStructure() {

this("Unknown", 0, "Unknown", true);

}

public MagicalStructure(String structureName) {

this(structureName, 50, "Unknown", true);

}

public MagicalStructure(String structureName, int magicPower, String location, boolean isActive) {

this.structureName = structureName;

this.magicPower = magicPower;

this.location = location;

this.isActive = isActive;

}

public abstract void castMagicSpell();

}

static class WizardTower extends MagicalStructure {

private int spellCapacity;

private String[] knownSpells;

public WizardTower() {

this("Wizard Tower", 100, "Hilltop", true, 0, new String[]{});

}

public WizardTower(String[] spells) {

this("Wizard Tower", 150, "Hilltop", true, spells.length, spells);

}

public WizardTower(String name, int magicPower, String location, boolean isActive, int spellCapacity, String[] knownSpells) {

super(name, magicPower, location, isActive);

this.spellCapacity = spellCapacity;

this.knownSpells = knownSpells;

}

@Override

public void castMagicSpell() {

System.out.println(structureName + " casts a powerful spell!");

}

public void doubleSpellCapacity() {

spellCapacity \*= 2;

}

}

static class EnchantedCastle extends MagicalStructure {

private int defenseRating;

private boolean hasDrawbridge;

public EnchantedCastle() {

this("Castle", 80, "Valley", true, 100, false);

}

public EnchantedCastle(String name, int defenseRating) {

this(name, 120, "Valley", true, defenseRating, true);

}

public EnchantedCastle(String name, int magicPower, String location, boolean isActive, int defenseRating, boolean hasDrawbridge) {

super(name, magicPower, location, isActive);

this.defenseRating = defenseRating;

this.hasDrawbridge = hasDrawbridge;

}

@Override

public void castMagicSpell() {

System.out.println(structureName + " radiates defensive magic!");

}

public void tripleDefense() {

defenseRating \*= 3;

}

}

static class MysticLibrary extends MagicalStructure {

private int bookCount;

private String ancientLanguage;

public MysticLibrary() {

this("Library", 60, "Town", true, 10, "Latin");

}

public MysticLibrary(int books) {

this("Library", 90, "Town", true, books, "Greek");

}

public MysticLibrary(String name, int magicPower, String location, boolean isActive, int bookCount, String ancientLanguage) {

super(name, magicPower, location, isActive);

this.bookCount = bookCount;

this.ancientLanguage = ancientLanguage;

}

@Override

public void castMagicSpell() {

System.out.println(structureName + " whispers knowledge spells!");

}

}

static class DragonLair extends MagicalStructure {

private String dragonType;

private int treasureValue;

public DragonLair() {

this("Dragon Lair", 200, "Mountain", true, "Fire Dragon", 1000);

}

public DragonLair(String type) {

this("Dragon Lair", 250, "Mountain", true, type, 2000);

}

public DragonLair(String name, int magicPower, String location, boolean isActive, String dragonType, int treasureValue) {

super(name, magicPower, location, isActive);

this.dragonType = dragonType;

this.treasureValue = treasureValue;

}

@Override

public void castMagicSpell() {

System.out.println(structureName + " breathes " + dragonType + " fire!");

}

}

public static boolean canStructuresInteract(MagicalStructure s1, MagicalStructure s2) {

if ((s1 instanceof WizardTower && s2 instanceof MysticLibrary) || (s1 instanceof MysticLibrary && s2 instanceof WizardTower)) return true;

if ((s1 instanceof EnchantedCastle && s2 instanceof DragonLair) || (s1 instanceof DragonLair && s2 instanceof EnchantedCastle)) return true;

if (s1 instanceof WizardTower && s2 instanceof WizardTower) return true;

return false;

}

public static String performMagicBattle(MagicalStructure attacker, MagicalStructure defender) {

if (attacker.magicPower > defender.magicPower) return attacker.structureName + " wins!";

else if (attacker.magicPower < defender.magicPower) return defender.structureName + " wins!";

else return "It's a draw!";

}

public static int calculateKingdomMagicPower(MagicalStructure[] structures) {

int total = 0;

for (MagicalStructure s : structures) total += s.magicPower;

return total;

}

public static void applySpecialEffects(MagicalStructure[] structures) {

for (int i = 0; i < structures.length; i++) {

for (int j = i + 1; j < structures.length; j++) {

MagicalStructure s1 = structures[i];

MagicalStructure s2 = structures[j];

if (s1 instanceof WizardTower && s2 instanceof MysticLibrary) ((WizardTower)s1).doubleSpellCapacity();

if (s2 instanceof WizardTower && s1 instanceof MysticLibrary) ((WizardTower)s2).doubleSpellCapacity();

if (s1 instanceof EnchantedCastle && s2 instanceof DragonLair) ((EnchantedCastle)s1).tripleDefense();

if (s2 instanceof EnchantedCastle && s1 instanceof DragonLair) ((EnchantedCastle)s2).tripleDefense();

}

}

}

public static void categorizeStructures(MagicalStructure[] structures) {

for (MagicalStructure s : structures) {

if (s instanceof WizardTower) System.out.println(s.structureName + " is a Wizard Tower");

else if (s instanceof EnchantedCastle) System.out.println(s.structureName + " is an Enchanted Castle");

else if (s instanceof MysticLibrary) System.out.println(s.structureName + " is a Mystic Library");

else if (s instanceof DragonLair) System.out.println(s.structureName + " is a Dragon Lair");

}

}

public static void calculateTaxes(MagicalStructure[] structures) {

for (MagicalStructure s : structures) {

int tax = 0;

if (s instanceof WizardTower) tax = 100;

else if (s instanceof EnchantedCastle) tax = 200;

else if (s instanceof MysticLibrary) tax = 150;

else if (s instanceof DragonLair) tax = 300;

System.out.println(s.structureName + " pays tax: " + tax);

}

}

public static void determineSpecialization(MagicalStructure[] structures) {

int magic = 0, defense = 0;

for (MagicalStructure s : structures) {

if (s instanceof WizardTower || s instanceof MysticLibrary) magic += s.magicPower;

if (s instanceof EnchantedCastle || s instanceof DragonLair) defense += s.magicPower;

}

if (magic > defense) System.out.println("Kingdom is Magic-focused");

else if (defense > magic) System.out.println("Kingdom is Defense-focused");

else System.out.println("Kingdom is Balanced");

}

public static void main(String[] args) {

MagicalStructure[] structures = {

new WizardTower(new String[]{"Fireball", "Shield"}),

new EnchantedCastle("Royal Castle", 200),

new MysticLibrary(50),

new DragonLair("Ice Dragon"),

new WizardTower()

};

categorizeStructures(structures);

applySpecialEffects(structures);

System.out.println("Total Kingdom Magic Power: " + calculateKingdomMagicPower(structures));

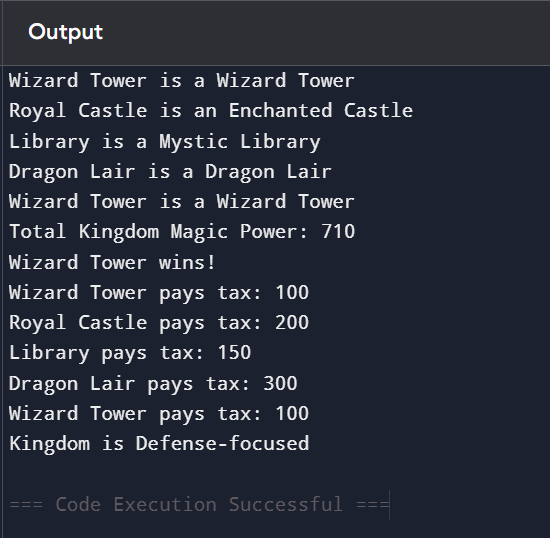
System.out.println(performMagicBattle(structures[0], structures[1]));

calculateTaxes(structures);

determineSpecialization(structures);

}

}



Q3. Space Station Crew Management System

Soln.

import java.util.\*;

public class SpaceStation {

public enum CrewRank {

CADET(1), OFFICER(2), COMMANDER(3), CAPTAIN(4), ADMIRAL(5);

private final int level;

CrewRank(int level) { this.level = level; }

public int getLevel() { return level; }

public boolean higherThan(CrewRank other) { return this.level > other.level; }

}

static class SpaceCrew {

private final String crewId;

private final String homeplanet;

private final CrewRank initialRank;

protected CrewRank currentRank;

protected int skillLevel;

protected int missionCount;

protected int spaceHours;

public static final String STATION\_NAME = "Stellar Odyssey";

public static final int MAX\_CREW\_CAPACITY = 50;

public SpaceCrew() {

this(UUID.randomUUID().toString(), getRandomPlanet(), CrewRank.CADET, CrewRank.CADET, 1, 0, 0);

}

public SpaceCrew(String homeplanet, CrewRank rank) {

this(UUID.randomUUID().toString(), homeplanet, rank, rank, 1, 0, 0);

}

public SpaceCrew(String homeplanet, CrewRank rank, int missions, int skills) {

this(UUID.randomUUID().toString(), homeplanet, rank, rank, skills, missions, missions \* 100);

}

public SpaceCrew(String crewId, String homeplanet, CrewRank initialRank, CrewRank currentRank, int skillLevel, int missionCount, int spaceHours) {

this.crewId = crewId;

this.homeplanet = homeplanet;

this.initialRank = initialRank;

this.currentRank = currentRank;

this.skillLevel = skillLevel;

this.missionCount = missionCount;

this.spaceHours = spaceHours;

}

public final String getCrewIdentification() {

return "ID: " + crewId + " | Planet: " + homeplanet + " | Initial Rank: " + initialRank;

}

public final boolean canBePromoted() {

return currentRank.getLevel() < CrewRank.ADMIRAL.getLevel() && skillLevel > 5 && missionCount > 3;

}

public final int calculateSpaceExperience() {

return missionCount \* 100 + spaceHours + skillLevel \* 10;

}

private static String getRandomPlanet() {

String[] planets = {"Earth", "Mars", "Venus", "Jupiter", "Saturn"};

return planets[new Random().nextInt(planets.length)];

}

}

static class PilotCrew extends SpaceCrew {

private final String flightCertification;

public PilotCrew(String homeplanet, CrewRank rank, String flightCertification) {

super(homeplanet, rank);

this.flightCertification = flightCertification;

}

public String getFlightCertification() {

return flightCertification;

}

}

static class ScienceCrew extends SpaceCrew {

private final String researchSpecialization;

public ScienceCrew(String homeplanet, CrewRank rank, String researchSpecialization) {

super(homeplanet, rank);

this.researchSpecialization = researchSpecialization;

}

public String getResearchSpecialization() {

return researchSpecialization;

}

}

static class EngineerCrew extends SpaceCrew {

private final String engineeringType;

public EngineerCrew(String homeplanet, CrewRank rank, String engineeringType) {

super(homeplanet, rank);

this.engineeringType = engineeringType;

}

public String getEngineeringType() {

return engineeringType;

}

}

public final static class SpaceStationRegistry {

private static final List<SpaceCrew> crewList = new ArrayList<>();

public static void addCrew(SpaceCrew crew) {

if (crewList.size() < SpaceCrew.MAX\_CREW\_CAPACITY) crewList.add(crew);

}

public static void printCrew() {

for (SpaceCrew c : crewList) {

System.out.println(c.getCrewIdentification() + " | Current Rank: " + c.currentRank);

}

}

public static void handleEmergency() {

boolean hasPilot = false, hasEngineer = false, hasScience = false;

for (SpaceCrew c : crewList) {

if (c instanceof PilotCrew) hasPilot = true;

if (c instanceof EngineerCrew) hasEngineer = true;

if (c instanceof ScienceCrew) hasScience = true;

}

if (hasPilot && hasEngineer && hasScience) System.out.println("Emergency handled successfully: all critical roles present!");

else System.out.println("Emergency failed: missing critical roles.");

}

public static void calculateStats() {

int totalExperience = 0;

for (SpaceCrew c : crewList) totalExperience += c.calculateSpaceExperience();

System.out.println("Total Space Experience: " + totalExperience);

}

}

public static void main(String[] args) {

PilotCrew p = new PilotCrew("Earth", CrewRank.OFFICER, "Advanced Flight Cert");

ScienceCrew s = new ScienceCrew("Mars", CrewRank.CADET, "Astrobiology");

EngineerCrew e = new EngineerCrew("Venus", CrewRank.COMMANDER, "Warp Systems");

SpaceStationRegistry.addCrew(p);

SpaceStationRegistry.addCrew(s);

SpaceStationRegistry.addCrew(e);

SpaceStationRegistry.printCrew();

SpaceStationRegistry.calculateStats();

SpaceStationRegistry.handleEmergency();

}

}

