**WEEK 5 – LAB PROBLEM**

Q1. Professional Virtual Pet Evolution System with Access Control

Soln.

import java.util.\*;

public class VirtualPetSystem {

public static final class PetSpecies {

private final String speciesName;

private final String[] evolutionStages;

private final int maxLifespan;

private final String habitat;

public PetSpecies(String speciesName, String[] evolutionStages, int maxLifespan, String habitat) {

if (speciesName == null || speciesName.isEmpty()) throw new IllegalArgumentException("Species name required");

if (evolutionStages == null || evolutionStages.length == 0) throw new IllegalArgumentException("Evolution stages required");

if (maxLifespan <= 0) throw new IllegalArgumentException("Max lifespan must be > 0");

if (habitat == null || habitat.isEmpty()) throw new IllegalArgumentException("Habitat required");

this.speciesName = speciesName;

this.evolutionStages = Arrays.copyOf(evolutionStages, evolutionStages.length);

this.maxLifespan = maxLifespan;

this.habitat = habitat;

}

public String getSpeciesName() { return speciesName; }

public String[] getEvolutionStages() { return Arrays.copyOf(evolutionStages, evolutionStages.length); }

public int getMaxLifespan() { return maxLifespan; }

public String getHabitat() { return habitat; }

@Override

public String toString() {

return "PetSpecies{" + "speciesName='" + speciesName + '\'' +

", maxLifespan=" + maxLifespan + ", habitat='" + habitat + '\'' + '}';

}

@Override

public int hashCode() { return Objects.hash(speciesName, maxLifespan, habitat); }

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof PetSpecies)) return false;

PetSpecies ps = (PetSpecies) o;

return maxLifespan == ps.maxLifespan &&

speciesName.equals(ps.speciesName) &&

habitat.equals(ps.habitat) &&

Arrays.equals(evolutionStages, ps.evolutionStages);

}

}

public static class VirtualPet {

private final String petId;

private final PetSpecies species;

private final long birthTimestamp;

private String petName;

private int age;

private int happiness;

private int health;

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

static final int MAX\_HAPPINESS = 100;

static final int MAX\_HEALTH = 100;

public static final String PET\_SYSTEM\_VERSION = "2.0";

private String currentStage;

public VirtualPet() {

this("Unknown", new PetSpecies("DefaultSpecies", DEFAULT\_EVOLUTION\_STAGES, 100, "Forest"), 0, 50, 50);

}

public VirtualPet(String petName) {

this(petName, new PetSpecies("DefaultSpecies", DEFAULT\_EVOLUTION\_STAGES, 100, "Forest"), 0, 60, 60);

}

public VirtualPet(String petName, PetSpecies species) {

this(petName, species, 0, 60, 60);

}

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

if (petName == null || petName.isEmpty()) throw new IllegalArgumentException("Pet name required");

this.petId = generatePetId();

this.species = Objects.requireNonNull(species, "Species required");

this.birthTimestamp = System.currentTimeMillis();

this.petName = petName;

setAge(age);

setHappiness(happiness);

setHealth(health);

updateEvolutionStage();

}

private String generatePetId() {

return UUID.randomUUID().toString();

}

private int validateStat(int value) {

if (value < 0) return 0;

if (value > 100) return 100;

return value;

}

private void updateEvolutionStage() {

String[] stages = species.getEvolutionStages();

int index = Math.min(age / 2, stages.length - 1);

currentStage = stages[index];

}

private void modifyHappiness(int delta) {

happiness = validateStat(happiness + delta);

}

private void modifyHealth(int delta) {

health = validateStat(health + delta);

}

protected int calculateFoodBonus(String foodType) {

if ("Meat".equalsIgnoreCase(foodType)) return 10;

if ("Fruit".equalsIgnoreCase(foodType)) return 7;

return 5;

}

protected int calculateGameEffect(String gameType) {

if ("Fetch".equalsIgnoreCase(gameType)) return 8;

if ("Puzzle".equalsIgnoreCase(gameType)) return 12;

return 5;

}

public void feedPet(String foodType) {

modifyHealth(calculateFoodBonus(foodType));

updateEvolutionStage();

}

public void playWithPet(String gameType) {

modifyHappiness(calculateGameEffect(gameType));

updateEvolutionStage();

}

void getInternalState() {

System.out.println("[DEBUG] " + toString());

}

public String getPetId() { return petId; }

public PetSpecies getSpecies() { return species; }

public long getBirthTimestamp() { return birthTimestamp; }

public String getPetName() { return petName; }

public void setPetName(String petName) { if (petName != null && !petName.isEmpty()) this.petName = petName; }

public int getAge() { return age; }

public void setAge(int age) { this.age = Math.max(0, age); updateEvolutionStage(); }

public int getHappiness() { return happiness; }

public void setHappiness(int happiness) { this.happiness = validateStat(happiness); }

public int getHealth() { return health; }

public void setHealth(int health) { this.health = validateStat(health); }

public String getCurrentStage() { return currentStage; }

@Override

public String toString() {

return "VirtualPet{" + "petId='" + petId + '\'' +

", species=" + species.getSpeciesName() +

", petName='" + petName + '\'' +

", age=" + age +

", happiness=" + happiness +

", health=" + health +

", stage='" + currentStage + '\'' + '}';

}

@Override

public int hashCode() { return Objects.hash(petId, species, petName); }

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof VirtualPet)) return false;

VirtualPet vp = (VirtualPet) o;

return petId.equals(vp.petId);

}

}

public static final class DragonPet {

private final String dragonType;

private final String breathWeapon;

private final VirtualPet basePet;

public DragonPet(String petName, String dragonType, String breathWeapon) {

this.dragonType = dragonType;

this.breathWeapon = breathWeapon;

this.basePet = new VirtualPet(petName, new PetSpecies("Dragon", new String[]{"Egg","Wyrmling","Young","Adult","Ancient"}, 200, "Cave"));

}

public VirtualPet getBasePet() { return basePet; }

public String getDragonType() { return dragonType; }

public String getBreathWeapon() { return breathWeapon; }

@Override

public String toString() {

return "DragonPet{" + "dragonType='" + dragonType + '\'' +

", breathWeapon='" + breathWeapon + '\'' +

", basePet=" + basePet + '}';

}

}

public static final class RobotPet {

private boolean needsCharging;

private int batteryLevel;

private final VirtualPet basePet;

public RobotPet(String petName) {

this.needsCharging = false;

this.batteryLevel = 100;

this.basePet = new VirtualPet(petName, new PetSpecies("Robot", new String[]{"Prototype","Model1","Model2","Advanced"}, 500, "Lab"));

}

public VirtualPet getBasePet() { return basePet; }

public boolean isNeedsCharging() { return needsCharging; }

public int getBatteryLevel() { return batteryLevel; }

public void useEnergy(int amount) {

batteryLevel -= amount;

if (batteryLevel <= 20) needsCharging = true;

}

public void charge() {

batteryLevel = 100;

needsCharging = false;

}

@Override

public String toString() {

return "RobotPet{" + "needsCharging=" + needsCharging +

", batteryLevel=" + batteryLevel +

", basePet=" + basePet + '}';

}

}

public static void main(String[] args) {

PetSpecies wolfSpecies = new PetSpecies("Wolf", new String[]{"Pup","Young","Adult","Elder"}, 80, "Mountains");

VirtualPet pet1 = new VirtualPet("Shadow", wolfSpecies, 1, 70, 90);

pet1.feedPet("Meat");

pet1.playWithPet("Fetch");

System.out.println(pet1);

DragonPet draco = new DragonPet("Draco","FireDragon","Flame Breath");

draco.getBasePet().playWithPet("Puzzle");

System.out.println(draco);

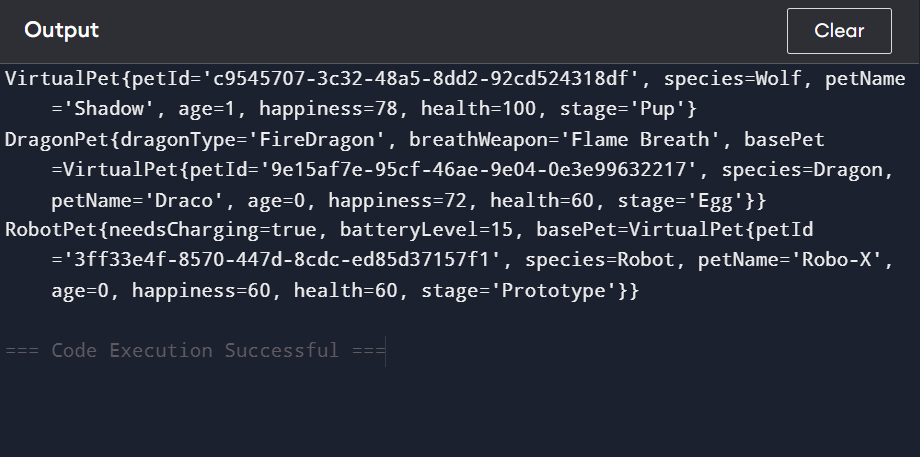
RobotPet robo = new RobotPet("Robo-X");

robo.useEnergy(85);

System.out.println(robo);

}

}



Q2. Medieval Kingdom Management with Security Architecture

Soln.

import java.util.\*;

public class KingdomSystem {

public static final class KingdomConfig {

private final String kingdomName;

private final int foundingYear;

private final String[] allowedStructureTypes;

private final Map<String,Integer> resourceLimits;

public KingdomConfig(String kingdomName, int foundingYear, String[] allowedStructureTypes, Map<String,Integer> resourceLimits) {

if (kingdomName == null || kingdomName.isEmpty()) throw new IllegalArgumentException("Kingdom name required");

if (foundingYear <= 0) throw new IllegalArgumentException("Invalid founding year");

if (allowedStructureTypes == null || allowedStructureTypes.length == 0) throw new IllegalArgumentException("Structure types required");

if (resourceLimits == null || resourceLimits.isEmpty()) throw new IllegalArgumentException("Resource limits required");

this.kingdomName = kingdomName;

this.foundingYear = foundingYear;

this.allowedStructureTypes = Arrays.copyOf(allowedStructureTypes, allowedStructureTypes.length);

this.resourceLimits = new HashMap<>(resourceLimits);

}

public String getKingdomName() { return kingdomName; }

public int getFoundingYear() { return foundingYear; }

public String[] getAllowedStructureTypes() { return Arrays.copyOf(allowedStructureTypes, allowedStructureTypes.length); }

public Map<String,Integer> getResourceLimits() { return new HashMap<>(resourceLimits); }

public static KingdomConfig createDefaultKingdom() {

return new KingdomConfig("Default Kingdom", 1000, new String[]{"Tower","Castle","Library","Lair"}, Map.of("Gold",1000,"Wood",500));

}

public static KingdomConfig createFromTemplate(String type) {

if ("Rich".equalsIgnoreCase(type)) {

return new KingdomConfig("Golden Empire", 1200, new String[]{"Tower","Castle","Library","Lair"}, Map.of("Gold",5000,"Wood",2000));

}

return createDefaultKingdom();

}

@Override

public String toString() {

return "KingdomConfig{"+"kingdomName='"+kingdomName+'\''+", foundingYear="+foundingYear+'}';

}

@Override

public int hashCode() { return Objects.hash(kingdomName,foundingYear); }

@Override

public boolean equals(Object o) {

if (this==o) return true;

if (!(o instanceof KingdomConfig)) return false;

KingdomConfig c=(KingdomConfig)o;

return foundingYear==c.foundingYear && kingdomName.equals(c.kingdomName);

}

}

public static class MagicalStructure {

private final String structureId;

private final long constructionTimestamp;

private final String structureName;

private final String location;

private int magicPower;

private boolean isActive;

private String currentMaintainer;

static final int MIN\_MAGIC\_POWER=0;

static final int MAX\_MAGIC\_POWER=1000;

public static final String MAGIC\_SYSTEM\_VERSION="3.0";

public MagicalStructure(String name,String location) {

this(name,location,100,true);

}

public MagicalStructure(String name,String location,int power) {

this(name,location,power,true);

}

public MagicalStructure(String name,String location,int power,boolean active) {

if(name==null||name.isEmpty()) throw new IllegalArgumentException("Name required");

if(location==null||location.isEmpty()) throw new IllegalArgumentException("Location required");

this.structureId=UUID.randomUUID().toString();

this.constructionTimestamp=System.currentTimeMillis();

this.structureName=name;

this.location=location;

setMagicPower(power);

this.isActive=active;

this.currentMaintainer="Unassigned";

}

public String getStructureId(){return structureId;}

public long getConstructionTimestamp(){return constructionTimestamp;}

public String getStructureName(){return structureName;}

public String getLocation(){return location;}

public int getMagicPower(){return magicPower;}

public void setMagicPower(int mp){ if(mp<MIN\_MAGIC\_POWER) mp=MIN\_MAGIC\_POWER; if(mp>MAX\_MAGIC\_POWER) mp=MAX\_MAGIC\_POWER; this.magicPower=mp;}

public boolean isActive(){return isActive;}

public void setActive(boolean active){this.isActive=active;}

public String getCurrentMaintainer(){return currentMaintainer;}

public void setCurrentMaintainer(String m){ if(m!=null&&!m.isEmpty()) this.currentMaintainer=m;}

@Override

public String toString(){return "MagicalStructure{"+structureName+", power="+magicPower+", active="+isActive+"}";}

}

public static class WizardTower {

private final int maxSpellCapacity;

private List<String> knownSpells;

private String currentWizard;

private final MagicalStructure base;

public WizardTower() {

this(50,new ArrayList<>(),"Unknown");

}

public WizardTower(int capacity,List<String> spells) {

this(capacity,spells,"Apprentice");

}

public WizardTower(int capacity,List<String> spells,String wizard) {

this.maxSpellCapacity=capacity;

this.knownSpells=new ArrayList<>(spells);

this.currentWizard=wizard;

this.base=new MagicalStructure("WizardTower","Hilltop");

}

public int getMaxSpellCapacity(){return maxSpellCapacity;}

public List<String> getKnownSpells(){return new ArrayList<>(knownSpells);}

public String getCurrentWizard(){return currentWizard;}

public void addSpell(String s){ if(knownSpells.size()<maxSpellCapacity) knownSpells.add(s);}

@Override

public String toString(){return "WizardTower{wizard="+currentWizard+", spells="+knownSpells+"}";}

}

public static class EnchantedCastle {

private final String castleType;

private int defenseRating;

private boolean hasDrawbridge;

private final MagicalStructure base;

public EnchantedCastle() {

this("Fort",100,false);

}

public EnchantedCastle(String type,int rating) {

this(type,rating,true);

}

public EnchantedCastle(String type,int rating,boolean drawbridge) {

this.castleType=type;

this.defenseRating=rating;

this.hasDrawbridge=drawbridge;

this.base=new MagicalStructure("EnchantedCastle","Valley");

}

public String getCastleType(){return castleType;}

public int getDefenseRating(){return defenseRating;}

public void setDefenseRating(int r){this.defenseRating=r;}

public boolean hasDrawbridge(){return hasDrawbridge;}

public void setDrawbridge(boolean d){this.hasDrawbridge=d;}

@Override

public String toString(){return "EnchantedCastle{type="+castleType+", defense="+defenseRating+"}";}

}

public static class MysticLibrary {

private final Map<String,String> bookCollection;

private int knowledgeLevel;

private final MagicalStructure base;

public MysticLibrary() {

this(new HashMap<>(),10);

}

public MysticLibrary(Map<String,String> books) {

this(books,50);

}

public MysticLibrary(Map<String,String> books,int knowledge) {

this.bookCollection=new HashMap<>(books);

this.knowledgeLevel=knowledge;

this.base=new MagicalStructure("MysticLibrary","Town");

}

public Map<String,String> getBookCollection(){return new HashMap<>(bookCollection);}

public int getKnowledgeLevel(){return knowledgeLevel;}

public void setKnowledgeLevel(int k){this.knowledgeLevel=k;}

@Override

public String toString(){return "MysticLibrary{knowledge="+knowledgeLevel+", books="+bookCollection.keySet()+"}";}

}

public static class DragonLair {

private final String dragonType;

private long treasureValue;

private int territorialRadius;

private final MagicalStructure base;

public DragonLair(String type) {

this(type,1000,50);

}

public DragonLair(String type,long treasure,int radius) {

this.dragonType=type;

this.treasureValue=treasure;

this.territorialRadius=radius;

this.base=new MagicalStructure("DragonLair","Mountain");

}

public String getDragonType(){return dragonType;}

public long getTreasureValue(){return treasureValue;}

public void setTreasureValue(long v){this.treasureValue=v;}

public int getTerritorialRadius(){return territorialRadius;}

public void setTerritorialRadius(int r){this.territorialRadius=r;}

@Override

public String toString(){return "DragonLair{dragonType="+dragonType+", treasure="+treasureValue+"}";}

}

public static class KingdomManager {

private final List<Object> structures;

private final KingdomConfig config;

public KingdomManager(KingdomConfig config) {

this.config=config;

this.structures=new ArrayList<>();

}

public void addStructure(Object s){structures.add(s);}

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

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

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

return false;

}

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

if(attacker instanceof WizardTower && defender instanceof EnchantedCastle) return "Spells vs Walls!";

if(attacker instanceof DragonLair && defender instanceof WizardTower) return "Dragon fire vs Wizard spells!";

return "Uneventful clash.";

}

public static int calculateKingdomPower(Object[] arr) {

int total=0;

for(Object o:arr) {

if(o instanceof WizardTower) total+=200;

if(o instanceof EnchantedCastle) total+=300;

if(o instanceof MysticLibrary) total+=150;

if(o instanceof DragonLair) total+=500;

}

return total;

}

private String determineStructureCategory(Object s) {

if(s instanceof WizardTower) return "Tower";

if(s instanceof EnchantedCastle) return "Castle";

if(s instanceof MysticLibrary) return "Library";

if(s instanceof DragonLair) return "Lair";

return "Unknown";

}

@Override

public String toString(){return "KingdomManager{config="+config+", structures="+structures+"}";}

}

public static void main(String[] args) {

KingdomConfig config=KingdomConfig.createDefaultKingdom();

KingdomManager manager=new KingdomManager(config);

WizardTower tower=new WizardTower(100,Arrays.asList("Fireball","Shield"),"Merlin");

EnchantedCastle castle=new EnchantedCastle("Royal",300,true);

MysticLibrary library=new MysticLibrary(Map.of("Grimoire","Ancient Magic"),80);

DragonLair lair=new DragonLair("Red Dragon",2000,100);

manager.addStructure(tower);

manager.addStructure(castle);

manager.addStructure(library);

manager.addStructure(lair);

System.out.println(config);

System.out.println(tower);

System.out.println(castle);

System.out.println(library);

System.out.println(lair);

System.out.println("Can interact: "+KingdomManager.canStructuresInteract(tower,library));

System.out.println("Battle: "+KingdomManager.performMagicBattle(lair,tower));

System.out.println("Kingdom Power: "+KingdomManager.calculateKingdomPower(new Object[]{tower,castle,library,lair}));

}

}

