WEEK 5 PRACTICE:

Name:Ramesh Harisabapathi Chettiar

Date of Submission:17/09/25

Ans1. import java.util.\*;

public class VirtualPetSystem {

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) {

this.speciesName = Objects.requireNonNull(speciesName);

this.evolutionStages = evolutionStages != null ? evolutionStages.clone() : new String[0];

this.maxLifespan = maxLifespan;

this.habitat = Objects.requireNonNull(habitat);

validateData();

}

private void validateData() {

if (speciesName.isEmpty() || maxLifespan <= 0) {

throw new IllegalArgumentException("Invalid species data");

}

}

public String getSpeciesName() { return speciesName; }

public String[] getEvolutionStages() { return evolutionStages.clone(); }

public int getMaxLifespan() { return maxLifespan; }

public String getHabitat() { return habitat; }

}

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", "Adult"};

static final int MAX\_HAPPINESS = 100;

static final int MAX\_HEALTH = 100;

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

public VirtualPet() {

this("Pet" + UUID.randomUUID().toString().substring(0, 5));

}

public VirtualPet(String petName) {

this(petName, new PetSpecies("Default", DEFAULT\_EVOLUTION\_STAGES, 100, "Forest"));

}

public VirtualPet(String petName, PetSpecies species) {

this.petId = generatePetId();

this.species = Objects.requireNonNull(species);

this.birthTimestamp = System.currentTimeMillis();

this.petName = Objects.requireNonNull(petName);

this.age = 0;

this.happiness = 50;

this.health = 100;

}

private String generatePetId() {

return "PET-" + UUID.randomUUID().toString().substring(0, 8);

}

public String getPetId() { return petId; }

public PetSpecies getSpecies() { return species; }

public long getBirthTimestamp() { return birthTimestamp; }

public String getPetName() { return petName; }

public int getAge() { return age; }

public int getHappiness() { return happiness; }

public int getHealth() { return health; }

public void setPetName(String petName) { this.petName = Objects.requireNonNull(petName); }

public void setHappiness(int happiness) {

this.happiness = Math.max(0, Math.min(MAX\_HAPPINESS, happiness));

}

public void setHealth(int health) {

this.health = Math.max(0, Math.min(MAX\_HEALTH, health));

}

public void feedPet(String foodType) {

modifyHappiness(10);

modifyHealth(5);

}

public void playWithPet(String gameType) {

modifyHappiness(15);

modifyHealth(-2);

}

private void modifyHappiness(int amount) {

setHappiness(happiness + amount);

}

private void modifyHealth(int amount) {

setHealth(health + amount);

}

@Override

public String toString() {

return "VirtualPet[name=" + petName + ", species=" + species.getSpeciesName() + "]";

}

}

static class DragonPet {

private final String dragonType;

private final String breathWeapon;

private VirtualPet basePet;

public DragonPet(String dragonType, String breathWeapon, VirtualPet basePet) {

this.dragonType = Objects.requireNonNull(dragonType);

this.breathWeapon = Objects.requireNonNull(breathWeapon);

this.basePet = Objects.requireNonNull(basePet);

}

public String getDragonType() { return dragonType; }

public String getBreathWeapon() { return breathWeapon; }

public VirtualPet getBasePet() { return basePet; }

}

static class RobotPet {

private boolean needsCharging;

private int batteryLevel;

private VirtualPet basePet;

public RobotPet(VirtualPet basePet) {

this.basePet = Objects.requireNonNull(basePet);

this.batteryLevel = 100;

this.needsCharging = false;

}

public boolean getNeedsCharging() { return needsCharging; }

public int getBatteryLevel() { return batteryLevel; }

public VirtualPet getBasePet() { return basePet; }

public void setBatteryLevel(int batteryLevel) {

this.batteryLevel = Math.max(0, Math.min(100, batteryLevel));

this.needsCharging = batteryLevel < 20;

}

}

public static void main(String[] args) {

PetSpecies dragonSpecies = new PetSpecies("Dragon", new String[]{"Egg", "Wyrmling", "Adult"}, 500, "Mountain");

VirtualPet myPet = new VirtualPet("Sparky", dragonSpecies);

System.out.println("Created pet: " + myPet);

System.out.println("Happiness: " + myPet.getHappiness());

myPet.feedPet("Dragon Fruit");

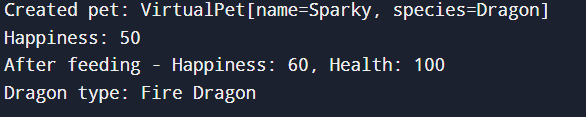
System.out.println("After feeding - Happiness: " + myPet.getHappiness() + ", Health: " + myPet.getHealth());

DragonPet dragon = new DragonPet("Fire Dragon", "Fire Breath", myPet);

System.out.println("Dragon type: " + dragon.getDragonType());

}

}



Ans 2. import java.util.\*;

public class MedievalKingdom {

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) {

this.kingdomName = Objects.requireNonNull(kingdomName);

this.foundingYear = foundingYear;

this.allowedStructureTypes = allowedStructureTypes != null ? allowedStructureTypes.clone() : new String[0];

this.resourceLimits = resourceLimits != null ? new HashMap<>(resourceLimits) : new HashMap<>();

validateConfig();

}

private void validateConfig() {

if (kingdomName.isEmpty() || foundingYear <= 0) {

throw new IllegalArgumentException("Invalid kingdom configuration");

}

}

public String getKingdomName() { return kingdomName; }

public int getFoundingYear() { return foundingYear; }

public String[] getAllowedStructureTypes() { return allowedStructureTypes.clone(); }

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

public static KingdomConfig createDefaultKingdom() {

return new KingdomConfig("Default Kingdom", 1000, new String[]{"Castle", "Tower"}, Map.of("Gold", 1000, "Wood", 5000));

}

}

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);

}

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

this(name, location, power, true);

}

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

this.structureId = "STRUCT-" + UUID.randomUUID().toString().substring(0, 8);

this.constructionTimestamp = System.currentTimeMillis();

this.structureName = Objects.requireNonNull(name);

this.location = Objects.requireNonNull(location);

setMagicPower(power);

this.isActive = active;

this.currentMaintainer = "Unknown";

}

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 boolean getIsActive() { return isActive; }

public String getCurrentMaintainer() { return currentMaintainer; }

public void setMagicPower(int magicPower) {

this.magicPower = Math.max(MIN\_MAGIC\_POWER, Math.min(MAX\_MAGIC\_POWER, magicPower));

}

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

public void setCurrentMaintainer(String currentMaintainer) { this.currentMaintainer = currentMaintainer; }

}

static class WizardTower {

private final int maxSpellCapacity;

private final List<String> knownSpells;

private String currentWizard;

private MagicalStructure baseStructure;

public WizardTower(String name, String location, int maxSpellCapacity) {

this.baseStructure = new MagicalStructure(name, location);

this.maxSpellCapacity = maxSpellCapacity;

this.knownSpells = new ArrayList<>();

this.currentWizard = "None";

}

public int getMaxSpellCapacity() { return maxSpellCapacity; }

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

public String getCurrentWizard() { return currentWizard; }

public MagicalStructure getBaseStructure() { return baseStructure; }

public void setCurrentWizard(String currentWizard) { this.currentWizard = currentWizard; }

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

}

static class KingdomManager {

private final List<Object> structures;

private final KingdomConfig config;

public KingdomManager(KingdomConfig config) {

this.config = Objects.requireNonNull(config);

this.structures = new ArrayList<>();

}

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

return (s1 instanceof WizardTower && s2 instanceof WizardTower) ||

(s1 instanceof MagicalStructure && s2 instanceof MagicalStructure);

}

public void addStructure(Object structure) {

structures.add(structure);

}

public List<Object> getStructures() { return new ArrayList<>(structures); }

public KingdomConfig getConfig() { return config; }

}

// Main method to run the program

public static void main(String[] args) {

// Create kingdom configuration

KingdomConfig config = KingdomConfig.createDefaultKingdom();

System.out.println("Kingdom: " + config.getKingdomName());

// Create kingdom manager

KingdomManager manager = new KingdomManager(config);

// Create structures

WizardTower tower = new WizardTower("Arcane Tower", "Northern Hills", 10);

MagicalStructure castle = new MagicalStructure("Royal Castle", "Central Plains", 500);

// Add structures to kingdom

manager.addStructure(tower);

manager.addStructure(castle);

// Test structure interaction

boolean canInteract = KingdomManager.canStructuresInteract(tower, castle);

System.out.println("Can structures interact: " + canInteract);

// Add spells to wizard tower

tower.addSpell("Fireball");

tower.addSpell("Lightning Bolt");

System.out.println("Tower spells: " + tower.getKnownSpells());

}

}

