```
// Medieval Kingdom Builder with Magic System
// Topics Covered: instanceof Type Checking, Inheritance, Constructor
Chaining, this Keyword
// Theme: Build a magical kingdom where different structures have unique
powers!
// Requirements: Create a kingdom building system with magical structures
that interact with
// each other.
// Hints: a. Create abstract MagicalStructure base class:
// • Fields: String structureName, int magicPower, String location,
boolean
// isActive
// • Constructor overloading with this() chaining
// • Abstract method: castMagicSpell()
// b. Create derived magical structures:
// • WizardTower (additional: int spellCapacity, String[] knownSpells)
// • EnchantedCastle (additional: int defenseRating, boolean
hasDrawbridge)
// • MysticLibrary (additional: int bookCount, String ancientLanguage)
// • DragonLair (additional: String dragonType, int treasureValue)
// c. Each structure type has unique constructor patterns:
// • WizardTower: Can be built empty, with basic spells, or fully equipped
// • Castle: Can be a simple fort, royal castle, or impregnable fortress
// • Library: Can start with few books or ancient collections
// • DragonLair: Different dragons have different lair requirements
// d. Implement magical interactions using instanceof:
// 4
// • static boolean canStructuresInteract(MagicalStructure s1,
// MagicalStructure s2)
// • static String performMagicBattle(MagicalStructure attacker,
// MagicalStructure defender)
// • static int calculateKingdomMagicPower(MagicalStructure[]
// structures)
// e. Twist: Some structure combinations create special effects:
// • WizardTower + Library = Knowledge boost (double spell capacity)
// • Castle + DragonLair = Dragon guard (triple defense)
// • Multiple towers = Magic network (shared spell pool)
// f. Create a KingdomManager that uses instanceof to:
// • Categorize structures by type
// • Calculate different tax rates for each structure type
```

```
// ● Determine kingdom specialization (Magic-focused, Defense-focused,
etc.)


import java.util.*;

abstract class MagicalStructure {
    protected String structureName;
    protected int magicPower;
    protected String location;
    protected boolean isActive;

    // Constructor chaining
    public MagicalStructure() {
        this("Unknown Structure", 50, "Unknown Realm", true);
    }

    public MagicalStructure(String structureName) {
        this(structureName, 75, "Central Kingdom", 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();
}

// WizardTower
class WizardTower extends MagicalStructure {
    private int spellCapacity;
    private String[] knownSpells;

    public WizardTower() {
        this("Basic Tower", 100, "Highlands", true, 3, new
String[]{"Fireball", "Shield", "Teleport"});
```

```
    }

    public WizardTower(String structureName, int spellCapacity) {
        this(structureName, 120, "Mystic Hill", true, spellCapacity, new
String[]{"Spark", "Heal"});
    }

    public WizardTower(String structureName, int magicPower, String
location, boolean isActive, int spellCapacity, String[] knownSpells) {
        super(structureName, magicPower, location, isActive);
        this.spellCapacity = spellCapacity;
        this.knownSpells = knownSpells;
    }

    @Override
    public void castMagicSpell() {
        System.out.println(structureName + " casts: " +
Arrays.toString(knownSpells));
    }

    public void doubleSpellCapacity() {
        this.spellCapacity *= 2;
    }

    public int getSpellCapacity() {
        return spellCapacity;
    }
}

// EnchantedCastle
class EnchantedCastle extends MagicalStructure {
    private int defenseRating;
    private boolean hasDrawbridge;

    public EnchantedCastle() {
        this("Simple Fort", 80, "Valley", true, 50, false);
    }

    public EnchantedCastle(String structureName, int defenseRating) {
```

```
        this(structureName, 90, "Royal Grounds", true, defenseRating,
true);
    }

    public EnchantedCastle(String structureName, int magicPower, String
location, boolean isActive, int defenseRating, boolean hasDrawbridge) {
        super(structureName, magicPower, location, isActive);
        this.defenseRating = defenseRating;
        this.hasDrawbridge = hasDrawbridge;
    }

    @Override
    public void castMagicSpell() {
        System.out.println(structureName + " activates magical shield with
defense " + defenseRating);
    }

    public void tripleDefense() {
        this.defenseRating *= 3;
    }

    public int getDefenseRating() {
        return defenseRating;
    }
}

// MysticLibrary
class MysticLibrary extends MagicalStructure {
    private int bookCount;
    private String ancientLanguage;

    public MysticLibrary() {
        this("Small Library", 60, "Forest Edge", true, 100, "Elvish");
    }

    public MysticLibrary(String structureName, int bookCount) {
        this(structureName, 70, "Arcane Woods", true, bookCount,
"Draconic");
    }
```

```
    public MysticLibrary(String structureName, int magicPower, String
location, boolean isActive, int bookCount, String ancientLanguage) {
        super(structureName, magicPower, location, isActive);
        this.bookCount = bookCount;
        this.ancientLanguage = ancientLanguage;
    }

    @Override
    public void castMagicSpell() {
        System.out.println(structureName + " reads ancient spell in " +
ancientLanguage);
    }
}

// DragonLair
class DragonLair extends MagicalStructure {
    private String dragonType;
    private int treasureValue;

    public DragonLair() {
        this("Cave of Ember", 150, "Volcano Ridge", true, "Fire Dragon",
1000);
    }

    public DragonLair(String dragonType, int treasureValue) {
        this("Lair of " + dragonType, 130, "Mountain Peak", true,
dragonType, treasureValue);
    }

    public DragonLair(String structureName, int magicPower, String
location, boolean isActive, String dragonType, int treasureValue) {
        super(structureName, magicPower, location, isActive);
        this.dragonType = dragonType;
        this.treasureValue = treasureValue;
    }

    @Override
    public void castMagicSpell() {
        System.out.println(dragonType + " breathes magic flames from " +
structureName);
```

```
    }
}

// KingdomManager
class KingdomManager {
    public static boolean canStructuresInteract(MagicalStructure s1,
MagicalStructure s2) {
        return s1.isActive && s2.isActive;
    }

    public static String performMagicBattle(MagicalStructure attacker,
MagicalStructure defender) {
        if (attacker.magicPower > defender.magicPower) {
            return attacker.structureName + " wins the battle!";
        } else {
            return defender.structureName + " defends successfully!";
        }
    }

    public static int calculateKingdomMagicPower(MagicalStructure[]
structures) {
        int total = 0;
        for (MagicalStructure s : structures) {
            total += s.magicPower;
        }
        return total;
    }

    public static void applySpecialEffects(MagicalStructure[] structures)
{
        boolean hasTower = false, hasLibrary = false, hasCastle = false,
hasLair = false;
        int towerCount = 0;

        for (MagicalStructure s : structures) {
            if (s instanceof WizardTower) {
                hasTower = true;
                towerCount++;
            }
            if (s instanceof MysticLibrary) hasLibrary = true;
```

```
            if (s instanceof EnchantedCastle) hasCastle = true;
            if (s instanceof DragonLair) hasLair = true;
        }

        for (MagicalStructure s : structures) {
            if (hasTower && hasLibrary && s instanceof WizardTower wt) {
                wt.doubleSpellCapacity();
                System.out.println("✨ Knowledge Boost applied to " +
wt.structureName);
            }
            if (hasCastle && hasLair && s instanceof EnchantedCastle ec) {
                ec.tripleDefense();
                System.out.println("🛡️ Dragon Guard applied to " +
ec.structureName);
            }
        }

        if (towerCount > 1) {
            System.out.println("🔗 Magic Network established among Wizard
Towers!");
        }
    }

    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 determineSpecialization(MagicalStructure[]
structures) {
        int magic = 0, defense = 0;
```

```
        for (MagicalStructure s : structures) {
            if (s instanceof WizardTower || s instanceof MysticLibrary)
magic++;
            if (s instanceof EnchantedCastle || s instanceof DragonLair)
defense++;
        }

        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.");
    }
}

// Main simulation
public class MagicalKingdomBuilder {
    public static void main(String[] args) {
        MagicalStructure[] kingdom = {
            new WizardTower(),
            new WizardTower("Arcane Spire", 5),
            new MysticLibrary("Grand Archive", 500),
            new EnchantedCastle("Royal Keep", 100),
            new DragonLair("Ice Dragon", 2000)
        };

        System.out.println("\n🔮 Kingdom Structures:");
        for (MagicalStructure s : kingdom) {
            s.castMagicSpell();
        }

        System.out.println("\n🧪 Magical Interactions:");
        KingdomManager.applySpecialEffects(kingdom);
        KingdomManager.categorizeStructures(kingdom);
        KingdomManager.determineSpecialization(kingdom);

        System.out.println("\n⚔️ Magic Battle:");
        System.out.println(KingdomManager.performMagicBattle(kingdom[0],
kingdom[3]));
```

```
        System.out.println("\n🔢 Total Magic Power: " +
KingdomManager.calculateKingdomMagicPower(kingdom));
    }
}
```

```
Compiling MagicalKingdomBuilder.java...
Compilation successful. Running program...


? Kingdom Structures:
Basic Tower casts: [Fireball, Shield, Teleport]
Arcane Spire casts: [Spark, Heal]
Grand Archive reads ancient spell in Draconic
Royal Keep activates magical shield with defense 100
Ice Dragon breathes magic flames from Lair of Ice Dragon

? Magical Interactions:
? Knowledge Boost applied to Basic Tower
? Knowledge Boost applied to Arcane Spire
?? Dragon Guard applied to Royal Keep
? Magic Network established among Wizard Towers!
Basic Tower is a Wizard Tower.
Arcane Spire is a Wizard Tower.
Grand Archive is a Mystic Library.
Royal Keep is an Enchanted Castle.
Lair of Ice Dragon is a Dragon Lair.
? Kingdom is Magic-focused.

?? Magic Battle:
Basic Tower wins the battle!

? Total Magic Power: 510

Program finished. Cleaning up...
MagicalKingdomBuilder.class file deleted successfully.
Press any key to continue . . .
```

```
// Virtual Pet Evolution Simulator
// Topics Covered: Constructor Overloading, this() Chaining, final
Keyword, static Usage
// Theme: Create a Tamagotchi-style virtual pet that evolves based on
care!
```

```java
// Requirements: Design a VirtualPet class that simulates pet evolution
through different life
// stages.
// Hints: a. Create VirtualPet class with fields:
// • final String petId (generated using UUID-like system)
// • String petName, String species, int age, int happiness, int health
// • static final String[] EVOLUTION_STAGES = {"Egg", "Baby", "Child",
// "Teen", "Adult", "Elder"}
// • static int totalPetsCreated
// b. Implement evolution-based constructors:
// • Default constructor: Creates a mysterious egg with random species
// • Constructor with name only: Pet starts as baby stage
// • Constructor with name and species: Pet starts as child stage
// • Full constructor: Specify all initial stats and stage
// c. Use this() chaining where all constructors eventually call the main
constructor
// d. Create unique methods:
// • evolvePet(): Changes evolution stage based on age and care
// • feedPet(), playWithPet(), healPet(): Affect happiness and health
// • simulateDay(): Ages pet and randomly affects stats
// 3
// • getPetStatus(): Returns current evolution stage
// • static generatePetId(): Creates unique IDs
// e. Twist: Pet dies if health reaches 0, becomes "Ghost" type that can't
evolve but can haunt
// other pets!
// f. In main method: Create a pet daycare with multiple pets, simulate
several days, show
// evolution progress



import java.util.*;

class VirtualPet {
    // Static and final fields
    private static final String[] EVOLUTION_STAGES = {"Egg", "Baby",
"Child", "Teen", "Adult", "Elder"};
    private static final Random rand = new Random();
    private static int totalPetsCreated = 0;
```

```
    // Instance fields
    private final String petId;
    private String petName;
    private String species;
    private int age;
    private int happiness;
    private int health;
    private int stageIndex;
    private boolean isGhost;

    // Default constructor
    public VirtualPet() {
        this("Mystery", getRandomSpecies(), 0, 50, 50, 0);
    }

    // Constructor with name only
    public VirtualPet(String petName) {
        this(petName, getRandomSpecies(), 0, 60, 60, 1);
    }

    // Constructor with name and species
    public VirtualPet(String petName, String species) {
        this(petName, species, 1, 70, 70, 2);
    }

    // Full constructor
    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;
        this.isGhost = false;
        totalPetsCreated++;
    }
```

```
    // Static method to generate unique pet ID
    public static String generatePetId() {
        return UUID.randomUUID().toString();
    }

    // Random species generator
    private static String getRandomSpecies() {
        String[] speciesList = {"Dragon", "Cat", "Alien", "Fox",
"Penguin"};
        return speciesList[rand.nextInt(speciesList.length)];
    }

    // Pet actions
    public void feedPet() {
        if (!isGhost) happiness += 10;
    }

    public void playWithPet() {
        if (!isGhost) happiness += 15;
    }

    public void healPet() {
        if (!isGhost) health += 20;
    }

    public void simulateDay() {
        if (isGhost) return;

        age++;
        happiness -= rand.nextInt(10);
        health -= rand.nextInt(15);

        if (health <= 0) {
            becomeGhost();
        } else {
            evolvePet();
        }
    }

    public void evolvePet() {
```

```
        if (isGhost) return;

        if (age > stageIndex * 2 && happiness > 40 && health > 30 &&
stageIndex < EVOLUTION_STAGES.length - 1) {
            stageIndex++;
        }
    }

    public String getPetStatus() {
        return isGhost ? "Ghost" : EVOLUTION_STAGES[stageIndex];
    }

    private void becomeGhost() {
        isGhost = true;
        species = "Ghost";
        happiness = 0;
        health = 0;
    }

    public void displayPet() {
        System.out.printf("🐾 PetID: %s | Name: %s | Species: %s | Age: %d
| Happiness: %d | Health: %d | Stage: %s%n",
                petId, petName, species, age, happiness, health,
getPetStatus());
    }

    public static int getTotalPetsCreated() {
        return totalPetsCreated;
    }
}

public class VirtualPetEvolutionSimulator {
    public static void main(String[] args) {
        List<VirtualPet> daycare = new ArrayList<>();
        daycare.add(new VirtualPet());
        daycare.add(new VirtualPet("Fluffy"));
        daycare.add(new VirtualPet("Bolt", "Fox"));
        daycare.add(new VirtualPet("Zara", "Dragon", 3, 80, 90, 3));

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

```
            System.out.println("\n🌞 Day " + day + " Simulation:");
            for (VirtualPet pet : daycare) {
                pet.simulateDay();
                pet.displayPet();
            }
        }

        System.out.println("\nTotal pets created: " +
VirtualPet.getTotalPetsCreated());
    }
}
```

```
? Day 1 Simulation:
? PetID: 896a8e43-b90e-4413-be6c-45004c501a0c | Name: Mystery | Species: Fox | Age: 1 | Happiness: 48 | Health: 36 | Stage: Baby
? PetID: 336381a2-43b3-451d-b7b9-dfa089fb3a1b | Name: Fluffy | Species: Alien | Age: 1 | Happiness: 51 | Health: 56 | Stage: Baby
? PetID: 53206acb-fbc0-4448-92e2-d56352a3327b | Name: Bolt | Species: Fox | Age: 2 | Happiness: 67 | Health: 68 | Stage: Child
? PetID: 4549e584-96c1-4e4d-92c2-b4348247aa16 | Name: Zara | Species: Dragon | Age: 4 | Happiness: 74 | Health: 90 | Stage: Teen

? Day 2 Simulation:
? PetID: 896a8e43-b90e-4413-be6c-45004c501a0c | Name: Mystery | Species: Fox | Age: 2 | Happiness: 46 | Health: 33 | Stage: Baby
? PetID: 336381a2-43b3-451d-b7b9-dfa089fb3a1b | Name: Fluffy | Species: Alien | Age: 2 | Happiness: 50 | Health: 45 | Stage: Baby
? PetID: 53206acb-fbc0-4448-92e2-d56352a3327b | Name: Bolt | Species: Fox | Age: 3 | Happiness: 61 | Health: 67 | Stage: Child
? PetID: 4549e584-96c1-4e4d-92c2-b4348247aa16 | Name: Zara | Species: Dragon | Age: 5 | Happiness: 73 | Health: 76 | Stage: Teen

? Day 3 Simulation:
? PetID: 896a8e43-b90e-4413-be6c-45004c501a0c | Name: Mystery | Species: Fox | Age: 3 | Happiness: 43 | Health: 31 | Stage: Child
? PetID: 336381a2-43b3-451d-b7b9-dfa089fb3a1b | Name: Fluffy | Species: Alien | Age: 3 | Happiness: 50 | Health: 41 | Stage: Child
? PetID: 53206acb-fbc0-4448-92e2-d56352a3327b | Name: Bolt | Species: Fox | Age: 4 | Happiness: 58 | Health: 67 | Stage: Child
? PetID: 4549e584-96c1-4e4d-92c2-b4348247aa16 | Name: Zara | Species: Dragon | Age: 6 | Happiness: 68 | Health: 68 | Stage: Teen

? Day 4 Simulation:
? PetID: 896a8e43-b90e-4413-be6c-45004c501a0c | Name: Mystery | Species: Fox | Age: 4 | Happiness: 42 | Health: 28 | Stage: Child
? PetID: 336381a2-43b3-451d-b7b9-dfa089fb3a1b | Name: Fluffy | Species: Alien | Age: 4 | Happiness: 46 | Health: 36 | Stage: Child
? PetID: 53206acb-fbc0-4448-92e2-d56352a3327b | Name: Bolt | Species: Fox | Age: 5 | Happiness: 50 | Health: 58 | Stage: Teen
? PetID: 4549e584-96c1-4e4d-92c2-b4348247aa16 | Name: Zara | Species: Dragon | Age: 7 | Happiness: 59 | Health: 68 | Stage: Adult

? Day 5 Simulation:
? PetID: 896a8e43-b90e-4413-be6c-45004c501a0c | Name: Mystery | Species: Fox | Age: 5 | Happiness: 34 | Health: 21 | Stage: Child
? PetID: 336381a2-43b3-451d-b7b9-dfa089fb3a1b | Name: Fluffy | Species: Alien | Age: 5 | Happiness: 46 | Health: 28 | Stage: Child
? PetID: 53206acb-fbc0-4448-92e2-d56352a3327b | Name: Bolt | Species: Fox | Age: 6 | Happiness: 44 | Health: 52 | Stage: Teen
? PetID: 4549e584-96c1-4e4d-92c2-b4348247aa16 | Name: Zara | Species: Dragon | Age: 8 | Happiness: 56 | Health: 57 | Stage: Adult

Total pets created: 4
```