COS20007: Object Oriented Programming

Pass Task 6.2: Key Object Oriented Concepts

Show Wai Yan/105293041

# Object-Oriented Programming: Four Key Principles

## Core Principles Explained

### 1. Encapsulation - Data Protection and Information Hiding

**Definition:** Encapsulation is the bundling of data (attributes) and methods that operate on that data within a single unit (class), while restricting direct access to internal components through access modifiers.

**Example:** In a BankAccount class, the balance field is marked as private, preventing external code from directly modifying it. Access is controlled through public methods like deposit() and withdraw() that include validation logic.

**Program Relation:** When creating shape drawing program, each Shape and its inheritance’s fields are encapsulated within the classes, accessible only through getter/setter methods that validated grade ranges (0-100).

### 2. Inheritance - Code Reusability and Hierarchical Relationships

**Definition:** Inheritance allows a new class (child/derived) to acquire properties and behaviors from an existing class (parent/base), promoting code reuse and establishing "is-a" relationships.

**Example:** A Vehicle base class contains common properties like speed and fuel. Car and Motorcycle classes inherit from Vehicle, gaining these properties while adding their specific features like numberOfDoors for Car.

**Program Relation:** In a shapes drawing program, a base Shape class provided common methods like Draw(), while specific shapes like Circle and Rectangle inherited these behaviors and implemented their own calculation logic.

### 3. Polymorphism - One Interface, Multiple Forms

**Definition:** Polymorphism enables objects of different classes to be treated as objects of a common base class, allowing the same interface to represent different underlying data types and behaviors.

**Example:** Different animal objects (Dog, Cat, Bird) all implement a makeSound() method differently, but can be stored in an Animal array and called uniformly, producing species-specific sounds.

**Program Relation:** In a shape drawing program, different Shape types (Circle, Rectangle, Line) all implemented a common Shape interface with a Draw(), isAt() and DrawOutline() methods, allowing each Shape to make various shapes through the same interface.

### 4. Abstraction - Hiding Complexity, Showing Essentials

**Definition:** Abstraction focuses on exposing only essential features of an object while hiding unnecessary implementation details, simplifying interaction with complex systems.

**Example:** A Car class provides simple methods like Start(), Accelerate(), and Brake() without exposing the internal complexities of engine ignition, fuel injection systems, or brake pad mechanics to the driver.

**Program Relation:** In shape drawing program, user do not need to know how each Shape is drawn to screen and how each Shape is drawn in a way. The SplashKit Library automatically refresh the screen and render our drawings. User only need to use SplashKit.RefreshScreen(), and do not need to know how it renders. For each Shape‘s isAt() method as well, users do not need to know how each Shape detects the area of click. So all the implementations are hide and user only need to know what they want to use and what method should they called.

### How These Principles Interconnect

These principles work synergistically to create robust, maintainable software:

* **Encapsulation** provides the foundation by creating secure, self-contained units
* **Inheritance** builds upon encapsulation by extending existing secure units
* **Polymorphism** leverages inheritance to create flexible, interchangeable components
* **Abstraction** simplifies the use of all these complex relationships

The beauty of OOP lies in how these principles reinforce each other: inheritance promotes code reuse while maintaining encapsulation, polymorphism enables flexible design while preserving abstraction, and abstraction makes complex inheritance hierarchies manageable.

## Exception Handling Examples

### 1. File Operations

try

{

string content = File.ReadAllText("data.txt");

// Process file content

}

catch (FileNotFoundException ex)

{

Console.WriteLine("File not found: " + ex.Message);

}

catch (UnauthorizedAccessException ex)

{

Console.WriteLine("Access denied: " + ex.Message);

}

File operations are inherently risky due to external dependencies (file existence, permissions, disk space).

### 2. Database Connections

try

{

using (SqlConnection conn = new SqlConnection(connectionString))

{

conn.Open();

// Database operations

}

}

catch (SqlException ex)

{

Console.WriteLine("Database connection failed: " + ex.Message);

}

catch (InvalidOperationException ex)

{

Console.WriteLine("Connection error: " + ex.Message);

}

Network issues, server downtime, or authentication failures can occur unpredictably in database operations.

### 3. User Input Validation

try

{

Console.Write("Enter your age: ");

string userInput = Console.ReadLine();

int age = int.Parse(userInput);

if (age < 0)

throw new ArgumentException("Age cannot be negative");

Console.WriteLine($"Your age is: {age}");

}

catch (FormatException ex)

{

Console.WriteLine("Please enter a valid number");

}

catch (ArgumentException ex)

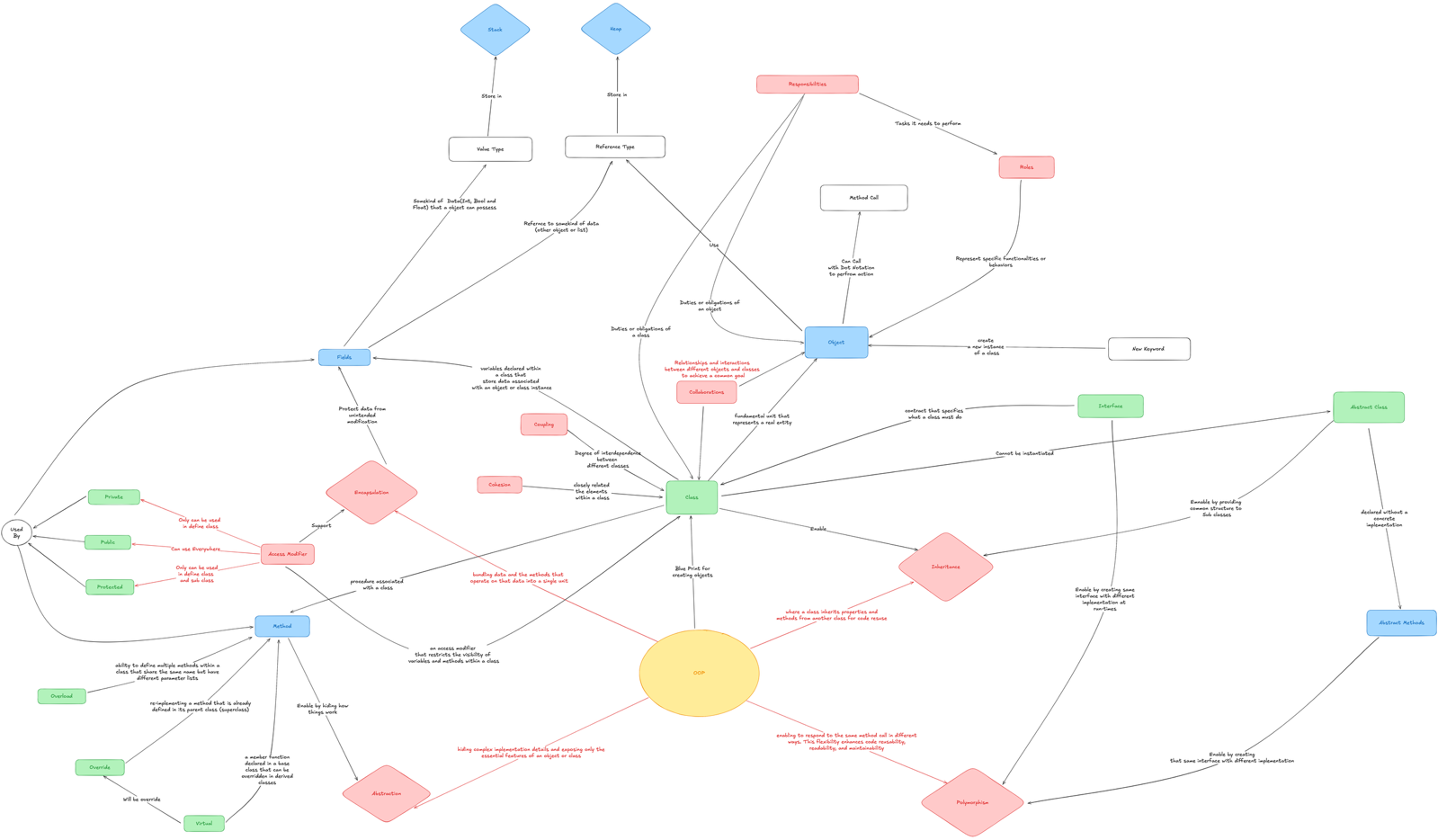
{

Console.WriteLine(ex.Message);

}

User input is unpredictable and may not match expected formats or constraints, requiring graceful error handling to maintain program stability.

# Concept Map



# References

Oracle Corporation. (2024). *The Java tutorials: Object-oriented programming concepts*. Oracle. https://docs.oracle.com/javase/tutorial/java/concepts/