Unit:3; Class:3

Encapsulation, Inheritance & Exception Handling

Instructor: Musfique Ahmed

Understanding Encapsulation

What is Encapsulation?

- Encapsulation **hides sensitive data** and only allows necessary access.
- Encapsulation = **Hiding data** + **Controlling access**
- Protects important information from accidental changes
- We use **private variables** and **getter/setter methods**

Real-World Example:

- **A Bank Account:** 💰
  - Your balance is **private** (you can't access it directly).
  - You use **methods like withdraw() and deposit()** to interact safely.

Practice Problem:

Create a **"Student"** class with a private variable `__grade`. Add a method `get_grade()` to return the grade safely.

Syntax of Encapsulation

```python
class BankAccount:
    def __init__(self, owner, balance):
        self.owner = owner
        self.__balance = balance  # Private variable

    def get_balance(self):  # Getter method
        return self.__balance

    def deposit(self, amount):  # Method to update private variable
        self.__balance += amount
```

Where & How to Use Encapsulation?

Protect sensitive data (e.g., passwords, bank balances)

- Control how data is accessed or modified

🚀 **Example: Video Game Character Stats (Health & Power are private!)**

```
class Player:
    def __init__(self, name, health):
        self.name = name
        self.__health = health

    def take_damage(self, damage):
        self.__health -= damage  # Private health modified safely
```

🛠️ **Practice Problem:**

Create a **"Student"** class with a private `__grade` and add getter/setter methods.

What is Inheritance?

- Inheritance = **Reuse code from a parent class**
- Child class **inherits** properties & methods from parent class
- Saves time & makes code cleaner!

📝 **Example:** A child inheriting eye color from parents! 👀

Syntax of Inheritance:

```
class Animal:  # Parent class
    def breathe(self):
        return "Breathing..."

class Dog(Animal):  # Child class inherits from Animal
    def bark(self):
        return "Woof! Woof!"

dog = Dog()
print(dog.breathe())  # ✅ Works because Dog inherits from Animal
```

Where & How to Use Inheritance?

✅ Create multiple similar objects without rewriting code
✅ Example: Different types of **Vehicles, Animals, or Bank Accounts**

🚗 **Example: Car inherits from Vehicle**

```
class Vehicle:
    def move(self):
        return "Moving..."

class Car(Vehicle):  # Car inherits from Vehicle
    def honk(self):
        return "Beep! Beep!"
```

🛠️ **Practice Problem:**
Create a **"Person"** class, then create a **"Student"** class that inherits from it.

What is Exception Handling?

- Catches **errors** and **prevents program crashes**
- Uses `try-except` to handle mistakes
- Helps programs **run smoothly**

📝 **Example:** Trying to divide by zero in a calculator!

Syntax of Exception Handling:

```
try:
    x = int(input("Enter a number: "))
    y = int(input("Enter another number: "))
    result = x / y  # Might cause error
except ZeroDivisionError:
    print("Oops! You cannot divide by zero.")
except ValueError:
    print("Please enter a valid number.")
finally:
    print("Program finished!")
```

Where & How to Use Exception Handling?

When expecting user input mistakes

- **When dealing with files or network requests**
- **Preventing crashes in games or apps**

🎮 **Example: Handling invalid input in a game**

```
try:
    age = int(input("Enter your age: "))
except ValueError:
    print("Please enter a number!")
```

🛠️ **Practice Problem:**

Write a Python program that asks for a number and catches `ValueError` if the input is not a number.

Practice Problem

🏦Simple Banking System:

Create a **Bank System** that:

1️⃣ Uses **Encapsulation** to store and protect account balances.
2️⃣ Uses **Inheritance** to create a `SavingsAccount` and `CheckingAccount` from a `BankAccount` class.
3️⃣ Uses **Exception Handling** to prevent invalid withdrawals.

Real-World Project Ideas

- **Password Manager (Encapsulation)** – Securely store and retrieve passwords
- **Game Character System (Inheritance)** – Different players with shared abilities
- **ATM Simulator (Exception Handling)** – Handle invalid transactions

🚀 Start coding & have fun! 🎉

Conclusion & Recap

✅ **Encapsulation** – Protects data
✅ **Inheritance** – Reuses code
✅ **Exception Handling** – Prevents crashes

- Keep practicing! 💡
- Ask questions! ❓
- Build projects! 🚀

Are you ready to code? 😊🎉

Thank You

Do the Quiz Please, you have 10 minutes to do that!