National University

Manila

College of Computing and Information Technologies

Computer Science Department

Programming Languages (CCPGLANG)

TERM 1 | AY2025-2026

RICO, RONALDO JR. DEL MUNDO

COM221

SUSAN S. CALUYA

Professor

**Title:** Write a program using abstraction.

1. ***Instruction Analysis***

   **Task**:

   The goal is to write a Java program that demonstrates abstraction, which is one of the fundamental ideas of object-oriented programming (OOP). The example uses an abstract class in which abstract methods are stated but not implemented, enabling subclasses to specify their own behavior.

   **Key Concepts**:

   - Abstraction: Hide implementation details and expose only the important functionalities.
   - Abstract Class: A class that cannot be instantiated and may include abstract methods.
   - Concrete Class: A subclass offers the actual implementation of abstract methods.

   **Expected Input and Output**:

   - Input: None (simulation only).
   - Output: Messages show the TV being switched on and off.

   **Implementation Required**:

   - Make two abstract methods (`turnOn()` and `turnOff()`) and an abstract class (`Abstract`).
   - Put into practice a specific subclass (`TVRemote`) that outlines the operation of these activities.
   - Demonstrate the concept in a main program.

2. ***Research Summary***

   A key idea in object-oriented programming (OOP) is abstraction, which allows programmers to concentrate on the functions of an object rather than its implementation. Either abstract classes or interfaces are used in Java to implement abstraction (Schildt, 2019). Subclasses handle the specific implementation, whereas abstract classes serve as a blueprint that specifies an object's structure.

   Because the abstract class specifies common behaviors that can be applied to many implementations, this method enables clearer and more maintainable code

(Horstmann & Cornell, 2024). An abstract class, for instance, can provide actions in a remote control system, such as turning a device on or off, but each individual device implements these actions in a unique way.

This structure increases scalability and decreases redundancy. Therefore, using abstraction is crucial for creating reliable and adaptable Java programs.

References:


[1] Horstmann, C. S. (2024). *Core java, volume I: fundamentals*. Pearson Education. Retrieved from https://ptgmedia.pearsoncmg.com/images/9780132354790/samplepages/0132354799_Sample.pdf

[2] Schildt, H. (2019). *Java: The Complete Reference* (11th ed.). McGraw-Hill Education. Retrieved from http://www.gandhicollegekada.org/department/Computer/E-Resources/Java-%20The%20Complete%20Reference,%20Eleventh%20Edition%20(%20PDFDrive.com%20).pdf

[3] GeeksforGeeks. (2025). *Abstraction in Java*. Retrieved from https://www.geeksforgeeks.org/java/abstraction-in-java-2/

[4] GeeksforGeeks. (2025). *Data Abstraction in Python*. Retrieved from https://www.geeksforgeeks.org/python/data-abstraction-in-python/

### 3. *Implementation, and Comparison*

**Java Implementation**:

```java
// Abstract class defining abstraction
abstract class Abstract {

    abstract void turnOn();
    abstract void turnOff();
}

// Concrete class implementing abstract methods
class TVRemote extends Abstract {

    @Override
    void turnOn() {
        System.out.println("TV is turned ON.");
    }

    @Override
    void turnOff() {
        System.out.println("TV is turned OFF.");
    }
}

// Main class to demonstrate abstraction
public class Main {

    public static void main(String[] args) {

        Geeks remote = new TVRemote();
        remote.turnOn();
        remote.turnOff();
    }
}
```

**Python Implementation**:

```
from abc import ABC, abstractmethod

# Abstract class defining abstraction
class Abstract(ABC):

    @abstractmethod
    def turn_on(self):
        pass

    @abstractmethod
    def turn_off(self):
        pass


# Concrete class implementing the abstract methods
class TVRemote(Abstract):

    def turn_on(self):
        print("TV is turned ON.")

    def turn_off(self):
        print("TV is turned OFF.")


# Main section to demonstrate abstraction
if __name__ == "__main__":
    remote = TVRemote()
    remote.turn_on()
    remote.turn_off()
```

## 4. ***Sample Simulations***

**Java Sample Output**:

```
TV is turned ON.
TV is turned OFF.
```

**Python Sample Output**:

```
TV is turned ON.
TV is turned OFF.
```

## 5. ***Reflection***

This project helped me grasp the notion of abstraction and how it simplifies program design by separating concept from implementation. Through this exercise, I learnt that abstract classes in Java describe the basic behavior of an object but allow the exact details to be handled by subclasses.

In the example, the abstract class Geeks specifies two methods, turnOn() and turnOff(), which the TVRemote class implements. This clearly shows how abstraction conceals the complexities of how the remote works while concentrating on what it accomplishes.

One problem I first experienced was knowing whether to utilize abstract classes instead of interfaces. However, I found that abstract classes are useful when we wish to give a shared basic structure or partial implementation for related objects.

Overall, this exercise enhanced my grasp of OOP concepts, notably abstraction, and how it helps code reusability, scalability, and clarity in software design.

**Evaluation Criteria**

| Criteria | Points | Your Score |
|---|---|---|
| Presentation | 15 | |
| Analysis of Instruction | 20 | |
| Depth and Accuracy of Research | 15 | |
| Correctness of Design/Simulation | 25 | |
| Clarity and Organization | 15 | |
| Reflection and Insight | 10 | |
| **Total** | 100 | |