Workshop: Building a Polymorphic Card Battle in C++

In this workshop, you will practice **runtime polymorphism** by building an abstract base class and two derived classes that work with a provided `main.cpp` battle simulation.

You will receive a `main.cpp` file that **must compile without changes to its logic**. Your job is to create the missing classes:

- `Card` (abstract base class)
- `WarriorCard` (derived class)
- `MageCard` (derived class)

Part 1 – Create the Class Files

Create **three header files** that define the class hierarchy:

1. `Card.h`
2. `WarriorCard.h`
3. `MageCard.h`

Create **three implementation files** that define their member functions:

1. `Card.cpp`
2. `WarriorCard.cpp`
3. `MageCard.cpp`

Requirements

- `Card` must be an **abstract base class**.
- `WarriorCard` and `MageCard` must **inherit publicly** from `Card`.
- The code in `main.cpp` must be able to store both derived objects inside an array of `Card*` and call functions through the base pointer.

Part 2 – Implement the Required Members

Your classes must provide the following **member variables** and **member functions** so they match what `main.cpp` calls.

Class: `Card` (Base / Abstract)

Member variables

- `protected std::string name_;`
- `protected int health_;`

Member functions

- **Constructor**
  - Initializes `name_` and `health_` using a member initialization list.
- **Virtual destructor** (required because you delete objects through `Card*`)
  - Prints: `"<name> is destroyed."`
- **getName()**
  - **Not virtual**
  - Returns `name_`
  - Should not be able to modify member variables
- **isAlive()**
  - **Not virtual**
  - Returns `true` if `health_ > 0`
  - Should not be able to modify member variables
- **attack()**
  - **Pure virtual** function
  - Must be overridden by `WarriorCard` and `MageCard`
  - Takes a reference to the card that is the target of the attack
  - Returns void
- **takeDamage**
  - **Virtual** function
  - Takes an integer as arguemnt to determine the amount of damage that will be taken
  - Returns void
  - Subtracts damage from `health_`, clamps to `0`, and prints:
    - `"<name> takes <amount> damage. (HP now <health>)"`

> Note: `attack()` must accept a **reference to a Card** (`Card&`) so that any derived card can be passed in.

Class: `WarriorCard` (Derived)

Member variables

- `private int strength_;`

Member functions

- **Constructor**
  - Calls the base constructor setting the card's name, and the amount of health to 50
  - Also initializes `strength_` to `3`
- **Destructor**

- **Overrides** the base destructor
- Prints a prefix: `"WarriorCard "`
  (so the final destruction output becomes: `WarriorCard Ragnar is destroyed.`)

- **attack()**
  - Overrides attack() from the base class
  - Prints: `"<name> (Warrior) swings at <targetName>!"`
  - Deals damage using:
    - `strength_ + rollDie()`
  - Calls:
    - `target.takeDamage(...)`

`WarriorCard` does **not** need to override `takeDamage()`.

Class: `MageCard` (Derived)

Member variables

- `private int spellPower_;`

Member functions

- **Constrcutor**
  - Calls the base constructor setting the card's name, and the amount of health to 30
  - Also initializes `spellPower_` to `2`

- **Destructor**
  - **Overrides** the base destructor
  - Prints a prefix: `"MageCard "`

- **attack()**
  - Overrides attack() from the base class
  - Prints: `"<name> (Mage) casts at <targetName>!"`
  - Deals damage using:
    - `spellPower_ * rollDie()`
  - Calls:
    - `target.takeDamage(...)`

- **takeDamage()**
  - **Overrides** the base implementation
  - Reduces incoming damage by `spellPower_` (a small shield):
    - `reduced = amount - spellPower_` (but never less than `0`)
  - Prints: `"<name> (Mage) reduces damage by <spellPower_>!"`
  - Calls the base version to apply damage + print the final HP line:
    - `Card::takeDamage(reduced)`

Part 3 – Testing

Use the **provided `main.cpp` file below** to test your classes. Your implementation must compile and run correctly with the following file **without modification**.

Part 4 – Build and Test

Run the program multiple times. Because it uses `rand()`, the battle outcome should vary.

Part 5 – Debugging Practice

Use a debugger (VS Studio, VS Code, or another IDE) and do the following:

1. Place a breakpoint on the line:
    - `cards[attacker]->attack(*cards[defender]);`
2. Step into the call and confirm that:
    - The correct derived `attack()` runs (Warrior or Mage)
3. When the defender is a mage, step into `takeDamage()` and confirm:
    - `MageCard::takeDamage()` runs first
    - it then calls `Card::takeDamage(reduced)`
4. Place a breakpoint on `delete cards[0];` and confirm:
    - the derived destructor runs first (`~WarriorCard()` / `~MageCard()`)
    - then the base destructor (`~Card()`)

Deliverables

Submit the following files:

- `Card.h` and `Card.cpp`
- `WarriorCard.h` and `WarriorCard.cpp`
- `MageCard.h` and `MageCard.cpp`