```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// 1. Classe base polimorfa astratta Component
class Component {
public:
    virtual ~Component() {}
    virtual void render() = 0; // Metodo virtuale puro
    virtual string getInfo() { return "Component"; } // Metodo virtuale
};

// 2. Classe astratta Container derivata da Component
class Container : public Component {
protected:
    vector<Component*> children;
public:
    virtual ~Container() {
        for (auto child : children) {
            delete child;
        }
    }

    virtual void add(Component* component) = 0; // Metodo virtuale puro

    void remove() { // Metodo non virtuale
        if (!children.empty()) {
            delete children.back();
            children.pop_back();
        }
    }

    string getInfo() override {
        return "Container with " + to_string(children.size()) + " children";
    }
};

// 3. Classe Panel derivata da Container
class Panel : public Container {
private:
    string layout;
public:
    Panel() : layout("default") {}

    void render() override {
        cout << "Rendering Panel with layout: " << layout << endl;
```

```
        for (auto child : children) {
            child->render();
        }
    }

    void add(Component* component) override {
        children.push_back(component);
    }

    void setLayout(string newLayout) {
        layout = newLayout;
    }

    string getInfo() override {
        return "Panel with layout: " + layout;
    }
};

// 4. Classe Button derivata da Component
class Button : public Component {
private:
    string label;
public:
    Button(const string& buttonLabel = "Button") : label(buttonLabel) {}

    void render() override {
        cout << "Rendering Button: " << label << endl;
    }

    void click() {
        cout << "Button " << label << " clicked" << endl;
    }

    string getInfo() override {
        return "Button: " + label;
    }
};

// 5. Classe ImageComponent con costruzione pubblica non permessa
class ImageComponent {
private:
    string imageUrl;

protected:
    // Costruttore protetto, permette solo la costruzione come sottooggetto
    ImageComponent(const string& url = "") : imageUrl(url) {}

public:
    virtual void resize() {
        cout << "Resizing image: " << imageUrl << endl;
```

```
    }

    string getImageUrl() const {
        return imageUrl;
    }

    void setImageUrl(const string& url) {
        imageUrl = url;
    }
};

// 6. Classe ImageButton derivata da Button e ImageComponent
class ImageButton : public Button, private ImageComponent {
public:
    ImageButton(const string& buttonLabel = "ImageButton")
        : Button(buttonLabel), ImageComponent() {}

    void render() override {
        cout << "Rendering ImageButton with image: " << getImageUrl() <<
endl;
    }

    void setImage(string url) {
        setImageUrl(url);
    }

    // Esporre selettivamente i metodi di ImageComponent
    void resize() {
        ImageComponent::resize();
    }

    string getInfo() override {
        return "ImageButton with image: " + getImageUrl();
    }
};

// 7. Classe Window derivata da Panel e ImageComponent con ereditarietà
virtuale
class Window : public Panel, public virtual ImageComponent {
private:
    string title;
public:
    Window(const string& windowTitle = "Window")
        : Component(), Panel(), ImageComponent("default_background.png"),
title(windowTitle) {}

    void render() override {
        cout << "Rendering Window: " << title << " with background: " <<
getImageUrl() << endl;
        Panel::render();
```

```
    }

    // Ridefinizione dell'operatore di assegnazione standard
    Window& operator=(const Window& other) {
        if (this != &other) {
            // Chiamare l'operatore di assegnazione delle classi base
            Panel::operator=(other);

            // Copiare i membri propri
            title = other.title;
            setImageUrl(other.getImageUrl());
        }
        return *this;
    }

    string getInfo() override {
        return "Window: " + title;
    }
};
```

Struttura della Gerarchia

La gerarchia implementata soddisfa tutti i requisiti specificati:

1. `Component` è la classe base polimorfa astratta alla radice con un metodo virtuale puro `render()` e un metodo virtuale `getInfo()`.
2. `Container` è una classe astratta derivata da `Component` che aggiunge un metodo virtuale puro `add(Component*)` e un metodo non virtuale `remove()`.
3. `Panel` è una classe concreta derivata da `Container` che implementa i metodi virtuali puri e aggiunge `setLayout(string)`.
4. `Button` è una classe concreta derivata da `Component` che implementa i metodi virtuali puri e aggiunge `click()`.
5. `ImageComponent` ha un costruttore protetto che impedisce la costruzione pubblica degli oggetti, permettendo solo la costruzione come sottooggetti. Ha un metodo virtuale `resize()`.
6. `ImageButton` deriva pubblicamente da `Button` e privatamente da `ImageComponent`. Aggiunge un metodo `setImage(string)`.
7. `Window` deriva pubblicamente da `Panel` e pubblicamente ma virtualmente da `ImageComponent`. Ridefinisce l'operatore di assegnazione con comportamento identico all'assegnazione standard.

L'implementazione gestisce correttamente l'ereditarietà virtuale, consentendo alla classe `Window` di ereditare da `ImageComponent` senza ambiguità, anche se `Panel` non deriva da `ImageComponent`.