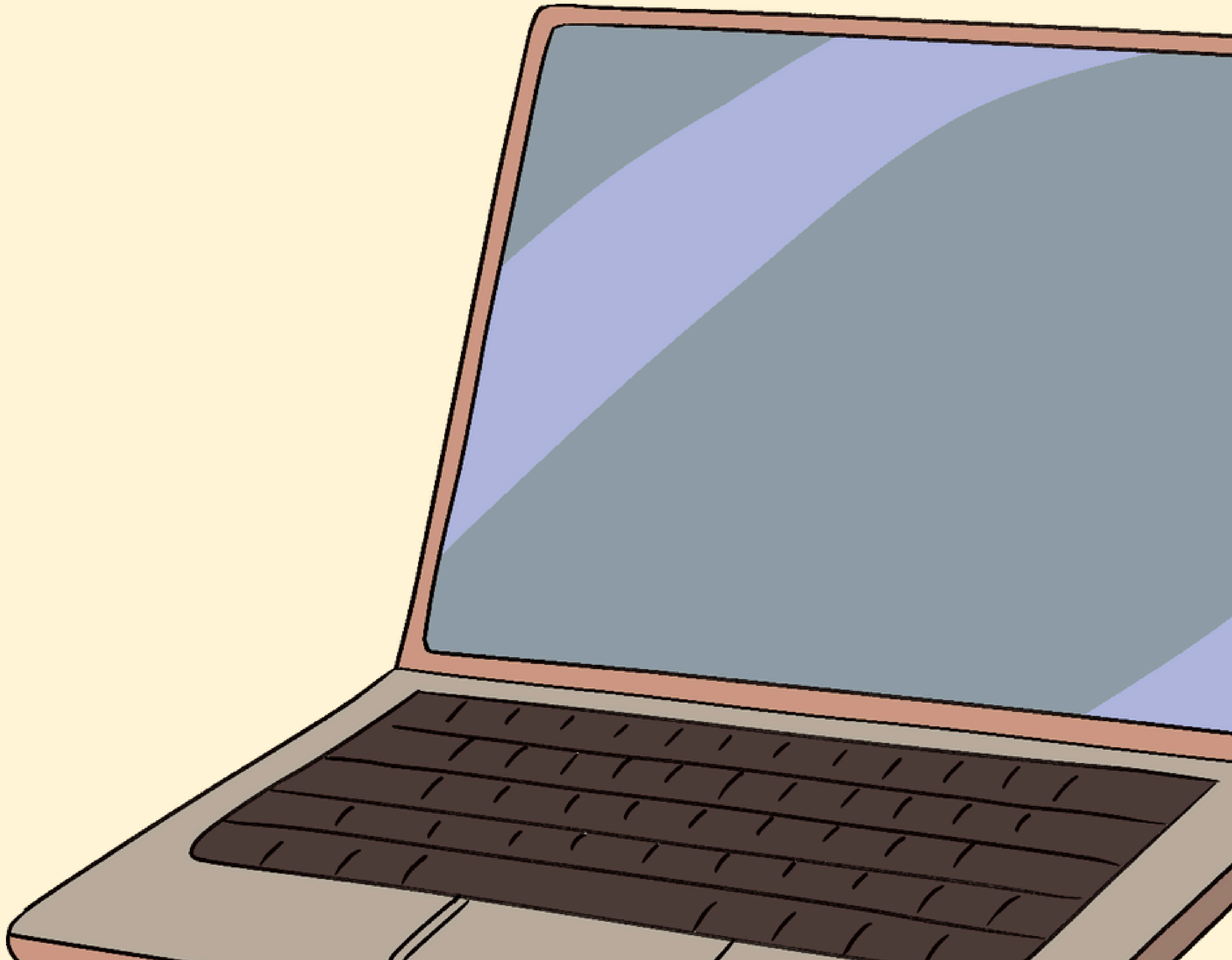
# BOOKWORM BOOK RECOMENDATION SYSTEM

Presented by:
Chuah Hui Wen A23CS0219
Chen Wei Jay Nickolas A23CS5028
Lim Xin Rou A23CS0240

# THE PROJECT DESCRIPTION

**Book Recommendation System**

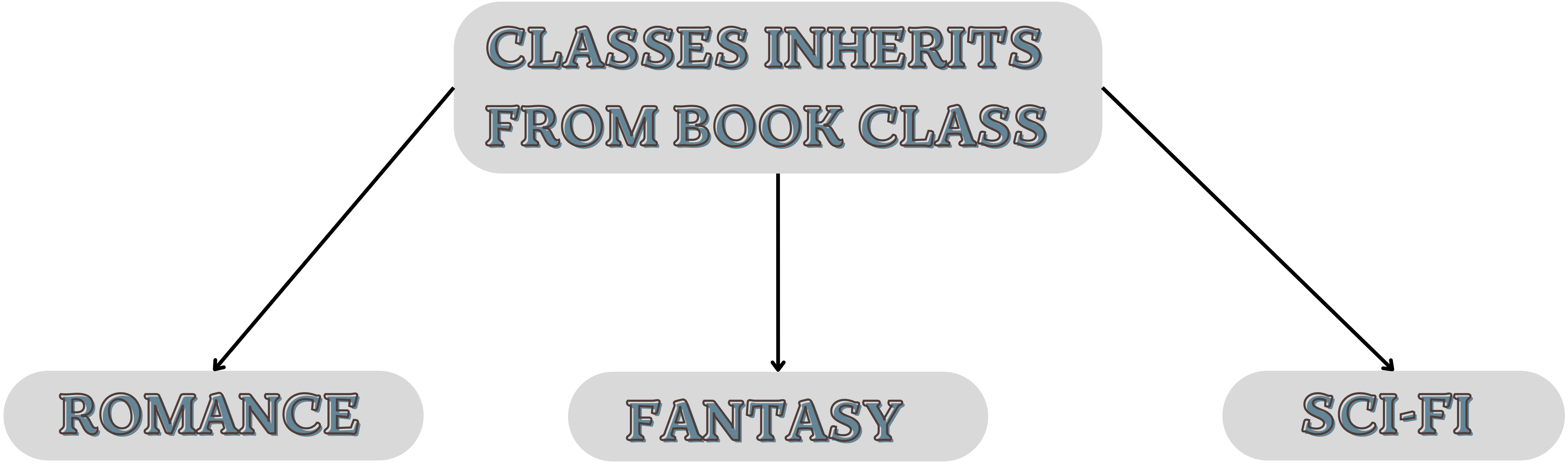
- Suggests books based on preferred genres.
- Help users discover new and interesting reads.
- Help people discover books they might otherwise miss.
- Saving time and effort.



## SYSTEM OBJECTIVE

- Provide personalized book recommendations to users
- Discover the books that match their taste
- Explore trending books and various genres
- Create and customize own book list
- Cultivate and spark interest of people to indulge in reading habits

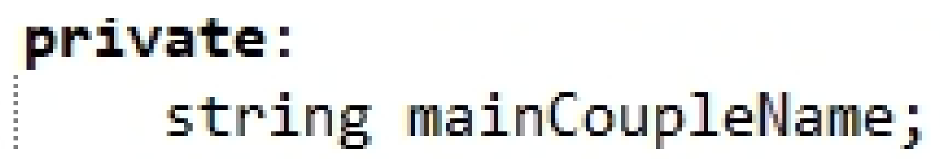
# INHERITANCE



# ROMANCE CLASS

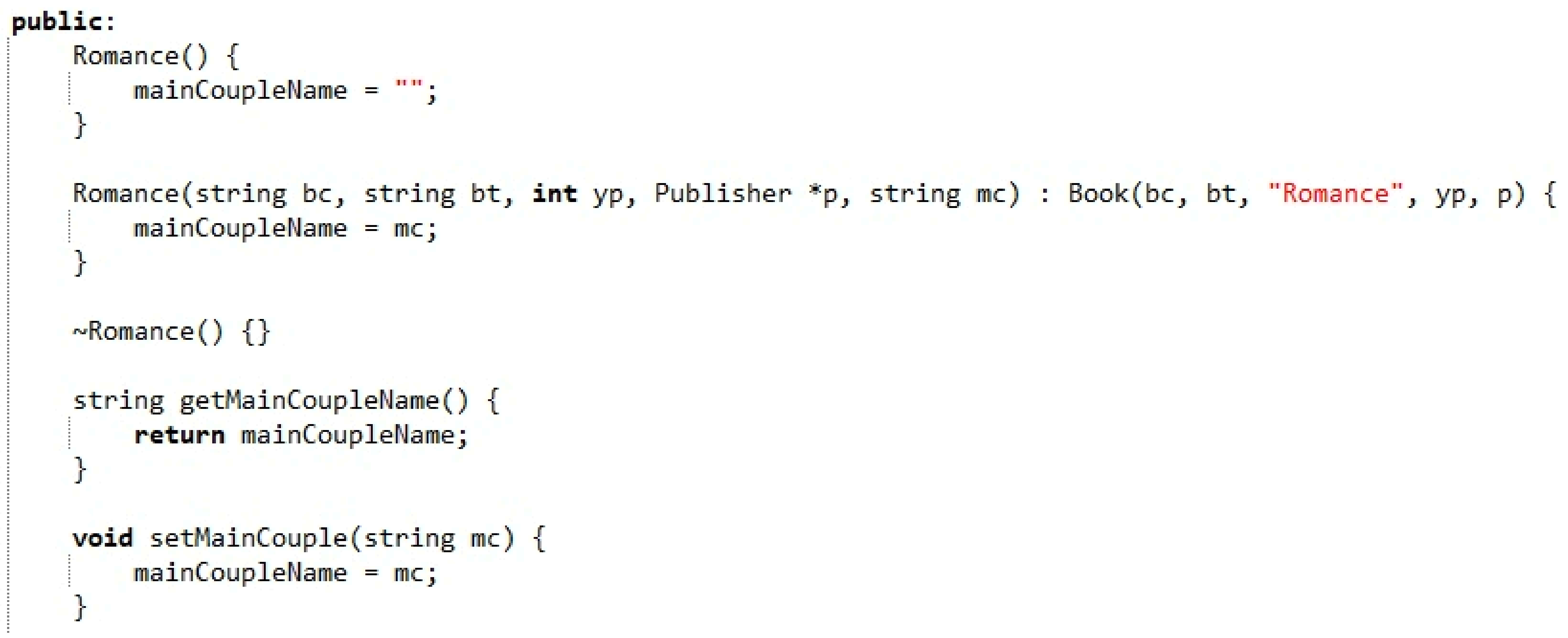
## ENCAPSULATION

### ATTRIBUTES



```
private:
    string mainCoupleName;
```

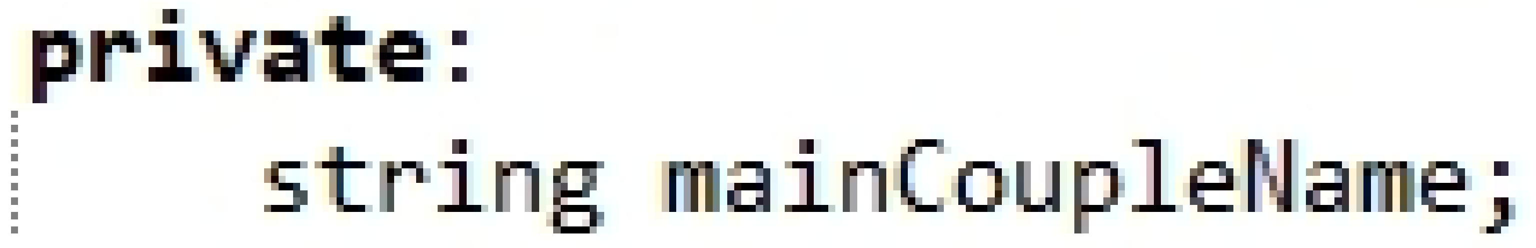
### METHODS



```
public:
    Romance() {
        mainCoupleName = "";
    }

    Romance(string bc, string bt, int yp, Publisher *p, string mc) : Book(bc, bt, "Romance", yp, p) {
        mainCoupleName = mc;
    }

    ~Romance() {}

    string getMainCoupleName() {
        return mainCoupleName;
    }

    void setMainCouple(string mc) {
        mainCoupleName = mc;
    }
```

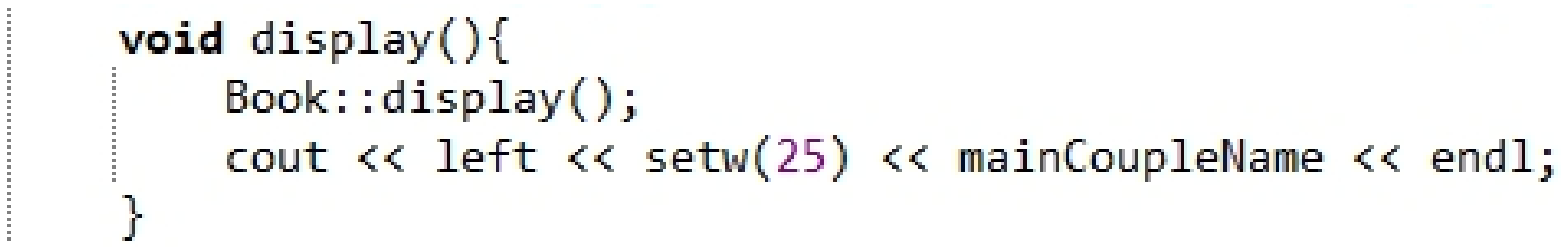
## INHERITANCE

- genre specific attributes



```
private:
    string mainCoupleName;
```

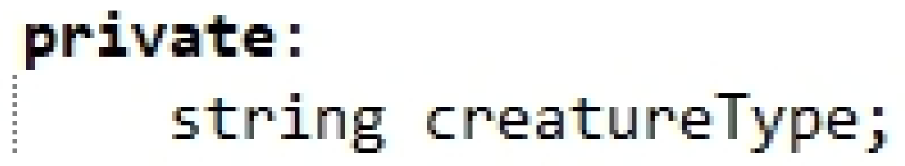
## POLYMORPHISM



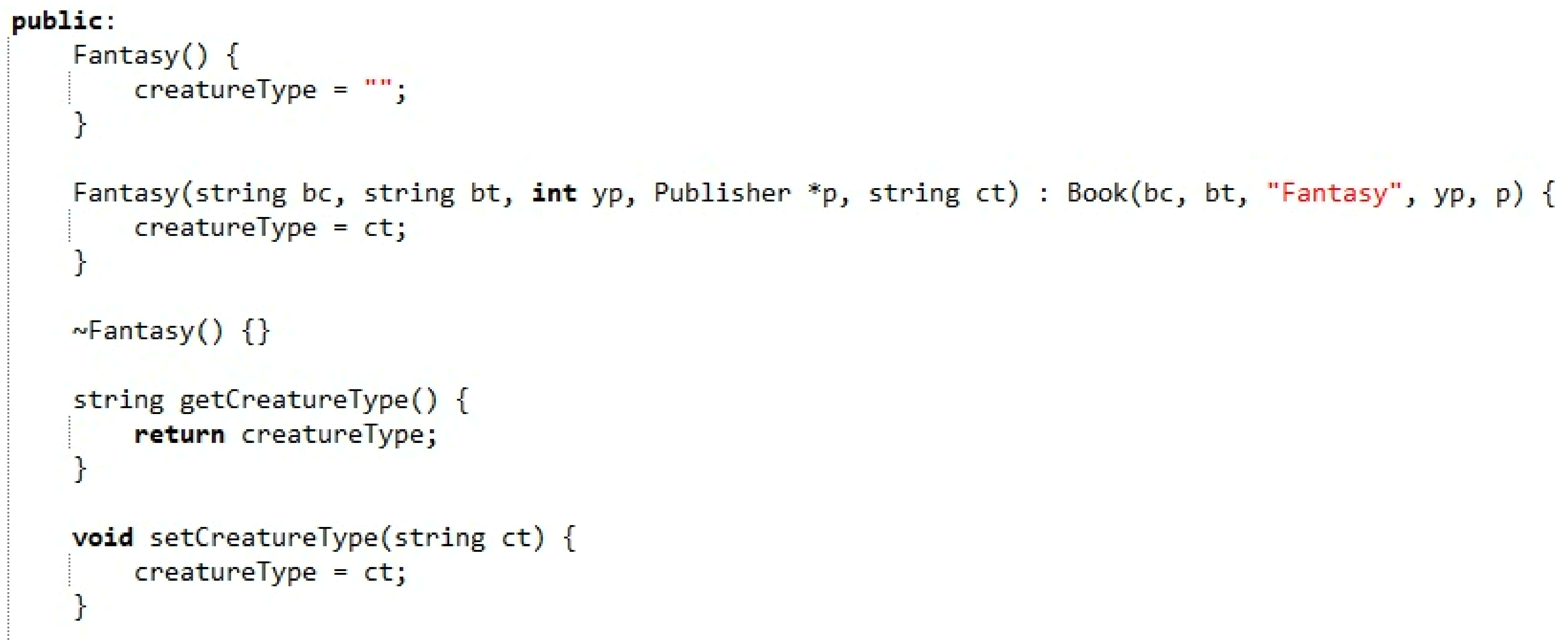
```
void display(){
    Book::display();
    cout << left << setw(25) << mainCoupleName << endl;
}
```

# FANTASY CLASS

$$\begin{tikzcd}
 & \text{ENCAPSULATION} \arrow[dl] \arrow[dr] & \\
\text{ATTRIBUTES} & & \text{METHODS}
\end{tikzcd}$$



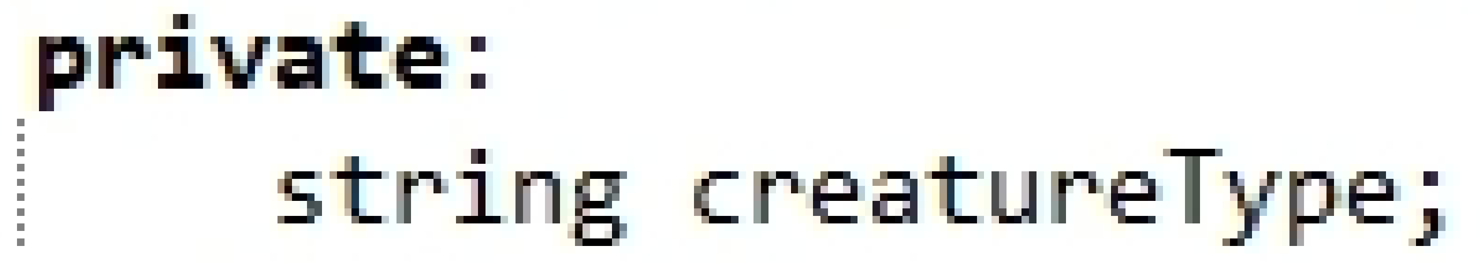
```
private:
    string creatureType;
```



```
public:
    Fantasy() {
        creatureType = "";
    }

    Fantasy(string bc, string bt, int yp, Publisher *p, string ct) : Book(bc, bt, "Fantasy", yp, p) {
        creatureType = ct;
    }

    ~Fantasy() {}

    string getCreatureType() {
        return creatureType;
    }

    void setCreatureType(string ct) {
        creatureType = ct;
    }
```

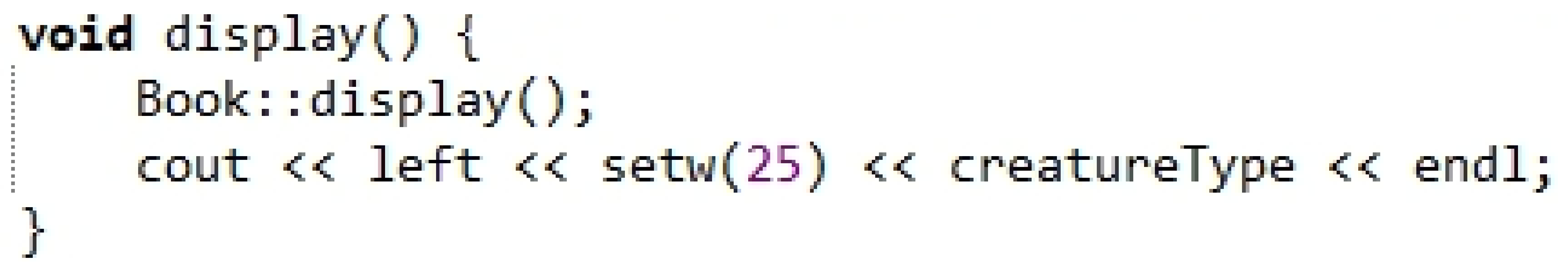
## INHERITANCE

- genre specific attributes



```
private:
    string creatureType;
```

## POLYMORPHISM

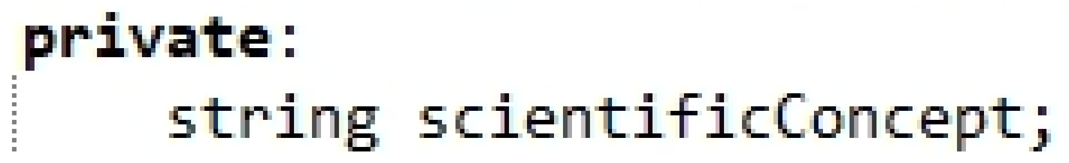


```
void display() {
    Book::display();
    cout << left << setw(25) << creatureType << endl;
}
```

# SCI-FI CLASS

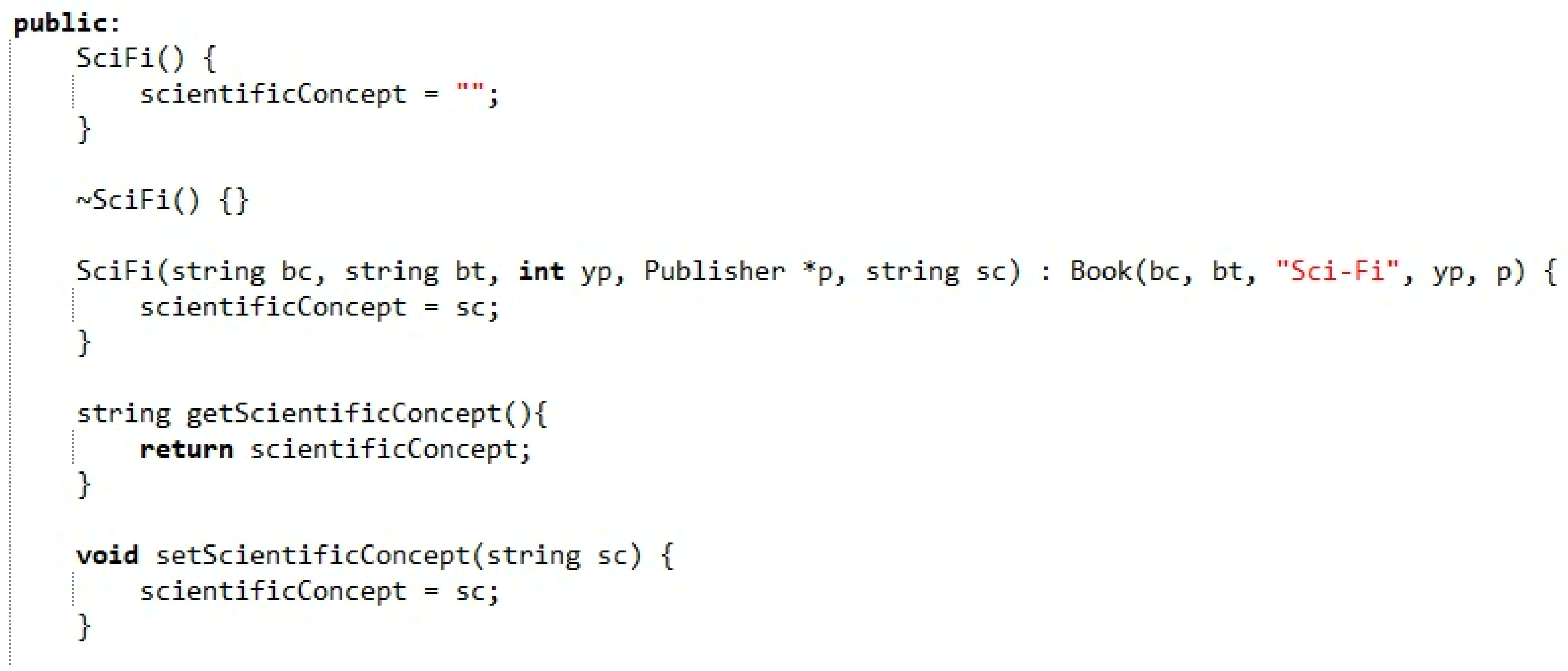
## ENCAPSULATION

### ATTRIBUTES



```
private:
    string scientificConcept;
```

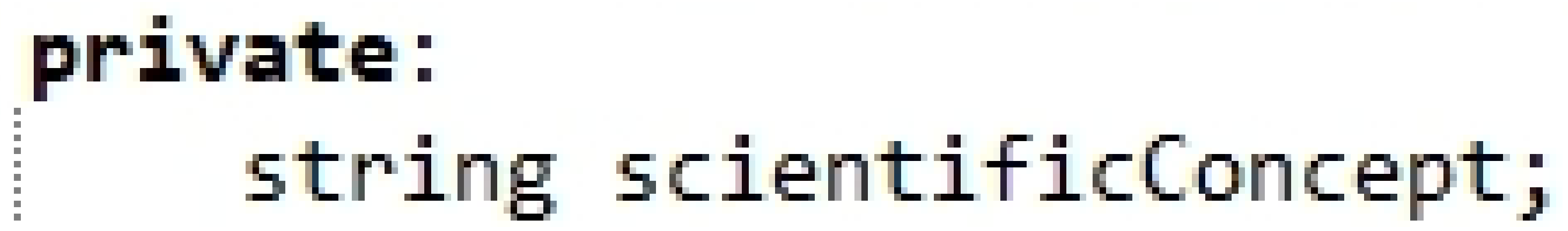
### METHODS



```
public:
    SciFi() {
        scientificConcept = "";
    }

    ~SciFi() {}

    SciFi(string bc, string bt, int yp, Publisher *p, string sc) : Book(bc, bt, "Sci-Fi", yp, p) {
        scientificConcept = sc;
    }

    string getScientificConcept(){
        return scientificConcept;
    }

    void setScientificConcept(string sc) {
        scientificConcept = sc;
    }
```

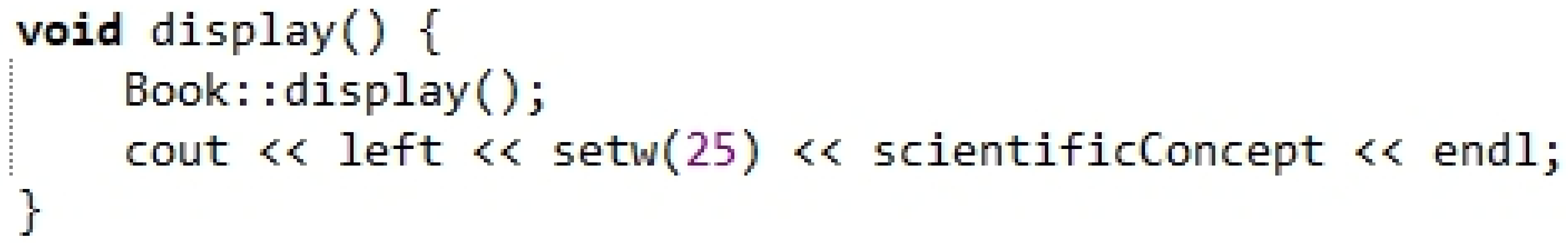
## INHERITANCE

- genre specific attributes



```
private:
    string scientificConcept;
```

## POLYMORPHISM



```
void display() {
    Book::display();
    cout << left << setw(25) << scientificConcept << endl;
}
```

# USER CLASS

**ENCAPSULATION**

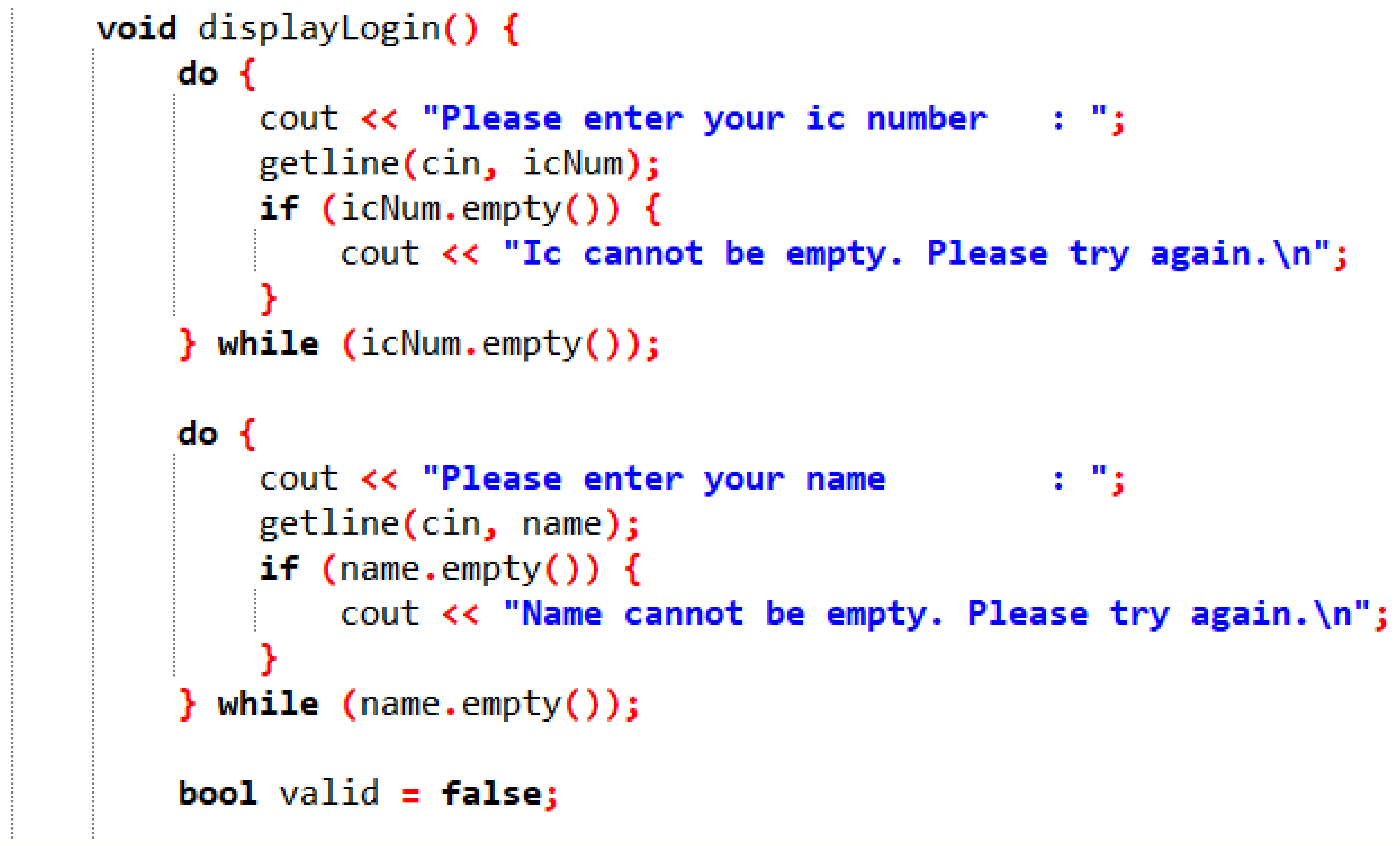
Attributes

```
class User{
    private:
        string name;
        string phoneNum;
        string icNum;
        Booklist booklist; // composition
```

Methods

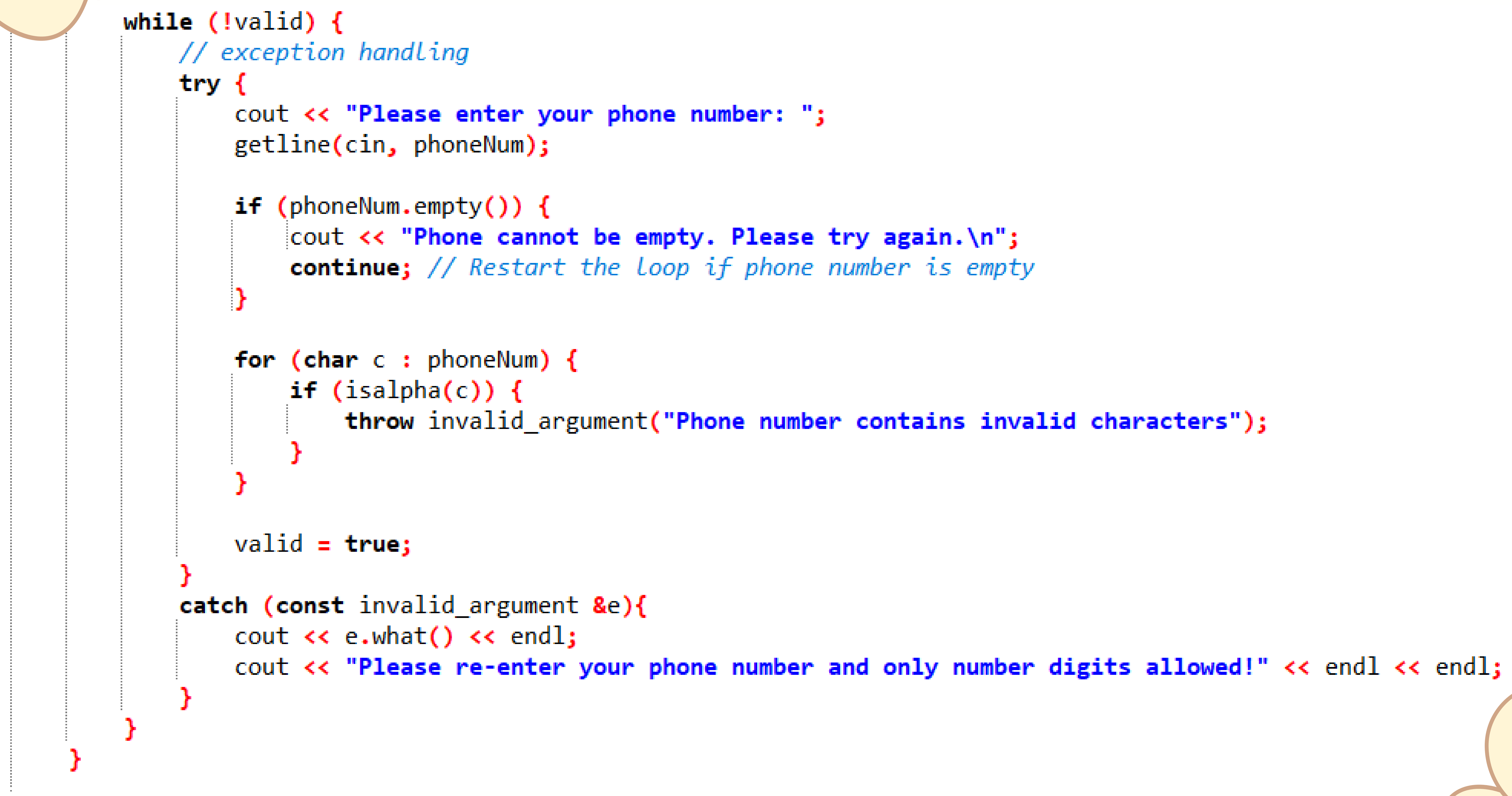
```
public:
    User(){}

    ~User(){}

    User(string ic, string n, string pn){
        icNum = ic;
        name = n;
        phoneNum = pn;
    }

    string getName() {
        return name;
    }

    string getPhoneNum() {
        return phoneNum;
    }
```

## SPECIAL METHODS



```
void displayLogin() {
    do {
        cout << "Please enter your ic number   : ";
        getline(cin, icNum);
        if (icNum.empty()) {
            cout << "Ic cannot be empty. Please try again.\n";
        }
    } while (icNum.empty());

    do {
        cout << "Please enter your name        : ";
        getline(cin, name);
        if (name.empty()) {
            cout << "Name cannot be empty. Please try again.\n";
        }
    } while (name.empty());

    bool valid = false;
```

## SPECIAL METHODS & EXCEPTION HANDLING



```cpp
    while (!valid) {
        // exception handling
        try {
            cout << "Please enter your phone number: ";
            getline(cin, phoneNum);

            if (phoneNum.empty()) {
                cout << "Phone cannot be empty. Please try again.\n";
                continue; // Restart the loop if phone number is empty
            }

            for (char c : phoneNum) {
                if (isalpha(c)) {
                    throw invalid_argument("Phone number contains invalid characters");
                }
            }

            valid = true;
        }
        catch (const invalid_argument &e){
            cout << e.what() << endl;
            cout << "Please re-enter your phone number and only number digits allowed!" << endl << endl;
        }
    }
}
```

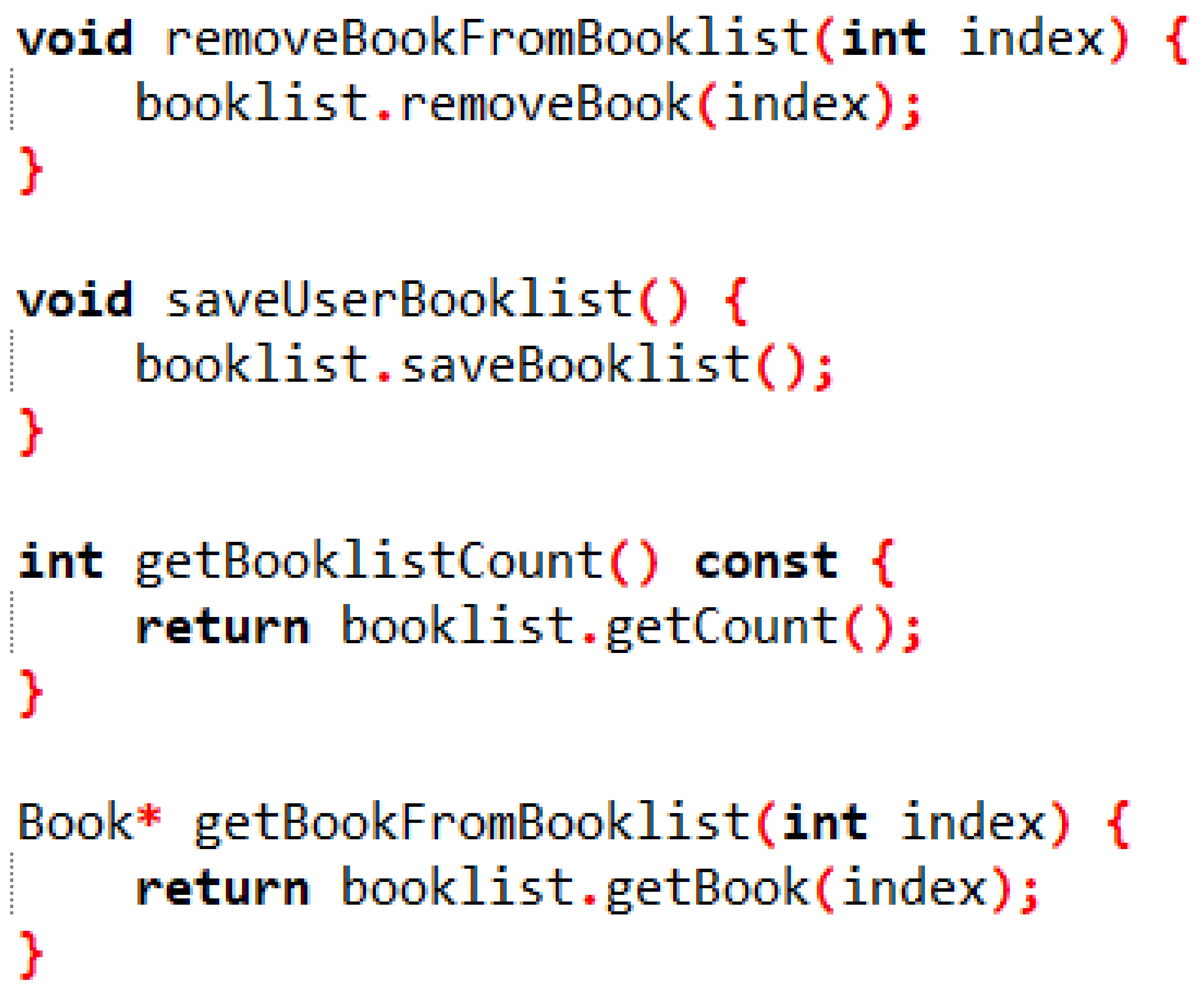
## SPECIAL METHODS

```cpp
void displayBooklist(){
    cout << "------------------------------------------------------------------------------------------------------------------\n";
    cout << "\t\t\t\t\t       User details   \t\t\t\t\t" << endl;
    cout << "------------------------------------------------------------------------------------------------------------------\n";
    cout << left << setw(20) << "Name " << ": " << name << endl;
    cout << left << setw(20) << "Phone Number " << ": " << phoneNum << endl;
    cout << left << setw(20) << "IC Number " << ": " << icNum << endl << endl;
    cout << "------------------------------------------------------------------------------------------------------------------\n";
    cout << "\t\t\t\t\t       " << name << "'s  booklist \t\t\t\t\t" << endl;
    cout << "------------------------------------------------------------------------------------------------------------------\n";
    booklist.display();

}
```

## SPECIAL METHODS

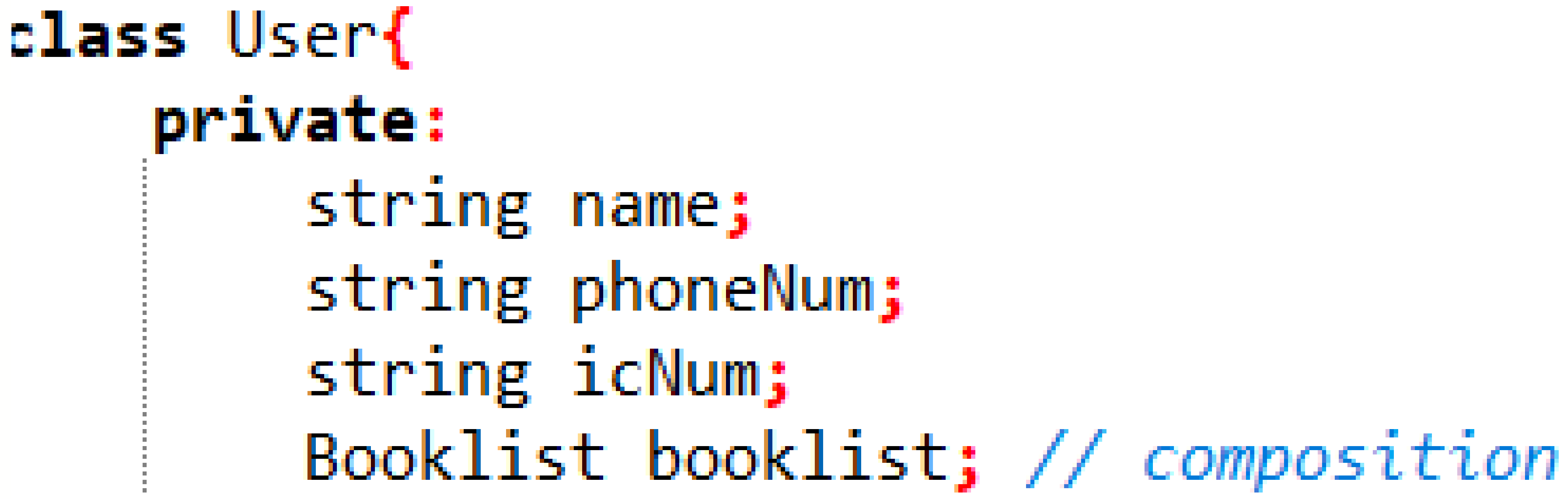
```
void addBookToBooklist(Book* book){
    int y = booklist.isBookInList(book);
    if(y==0)
        booklist.addBook(book);
    else if(y==1)
        cout << "This book is already in your personalized book list! :)" << endl << endl;

}
```

## SPECIAL METHODS



```
void removeBookFromBooklist(int index) {
    booklist.removeBook(index);
}

void saveUserBooklist() {
    booklist.saveBooklist();
}

int getBooklistCount() const {
    return booklist.getCount();
}

Book* getBookFromBooklist(int index) {
    return booklist.getBook(index);
}
```

## COMPOSITION



```
class User{
    private:
        string name;
        string phoneNum;
        string icNum;
        Booklist booklist; // composition
```

# BOOKLIST CLASS

**ENCAPSULATION**

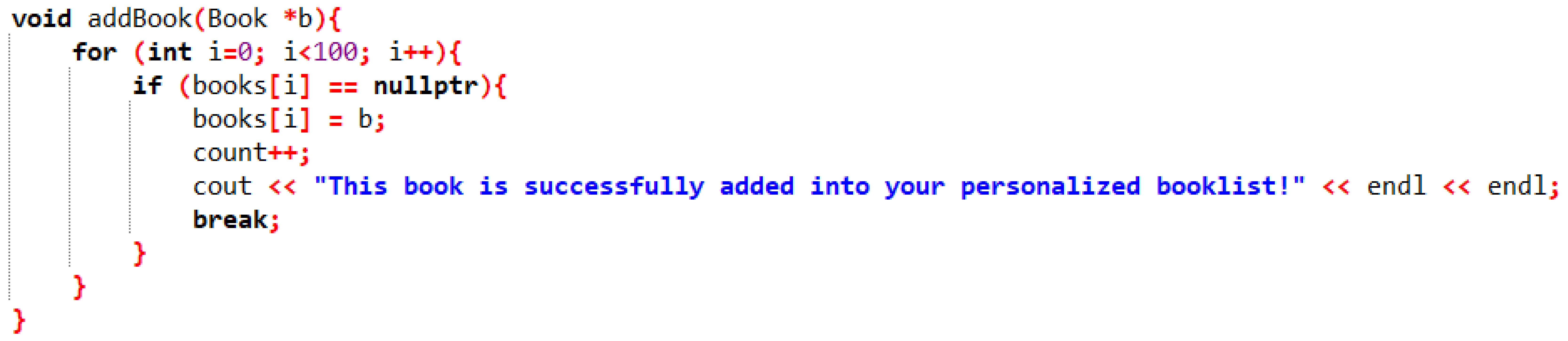
Attributes

Methods

```
class Booklist {
    private:
        Book* books[100];
        int count;
```

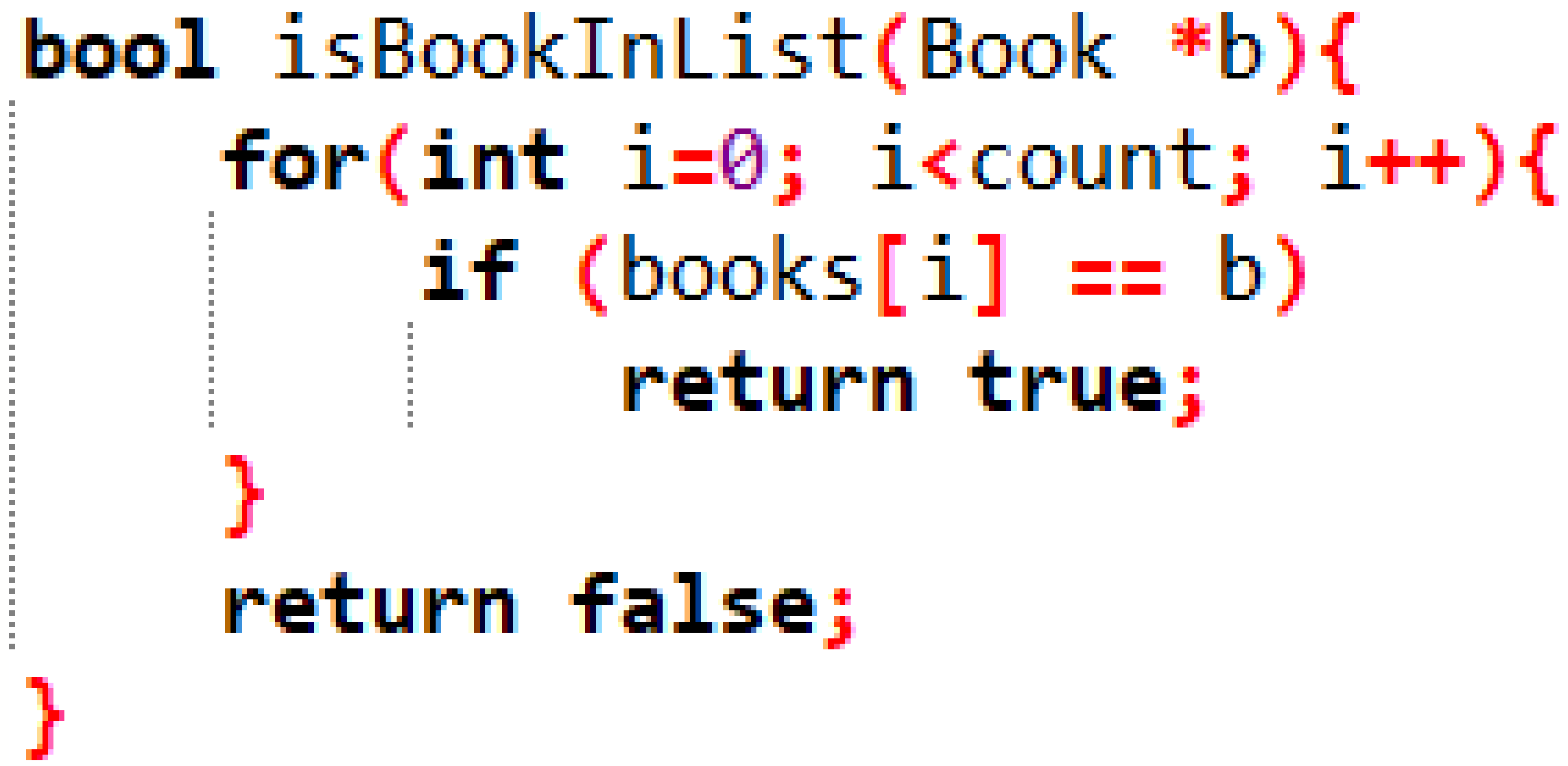
```
public:
    Booklist(){
        count = 0;
        for(int i=0; i<100; i++){
            books[i] = NULL;
        }
    }

    ~Booklist(){}

    int getCount(){ return count; }

    void setCount(int c){ count = c; }

    Book* getBook(int index){
        return books[index];
    }
```

## SPECIAL METHODS



```cpp
void addBook(Book *b){
    for (int i=0; i<100; i++){
        if (books[i] == nullptr){
            books[i] = b;
            count++;
            cout << "This book is successfully added into your personalized booklist!" << endl << endl;
            break;
        }
    }
}
```

## SPECIAL METHODS



```
bool isBookInList(Book *b){
    for(int i=0; i<count; i++){
        if (books[i] == b)
            return true;
    }
    return false;
}
```

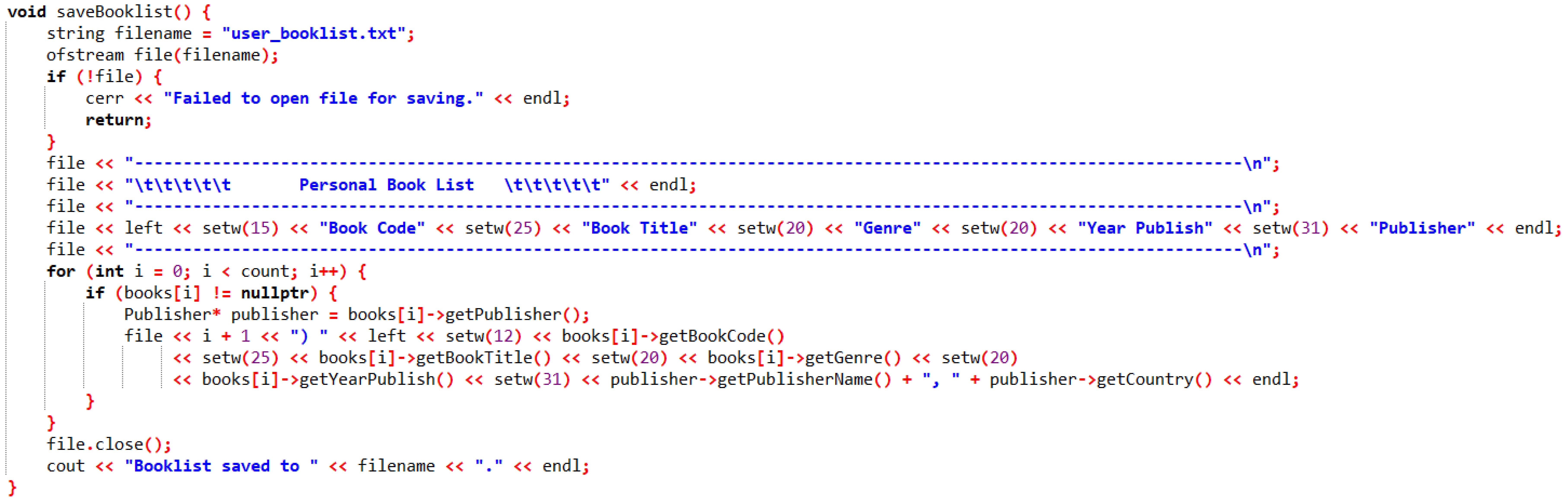
## SPECIAL METHODS

```
void display(){
    cout << left << setw(11) << "Book Code" << setw(25) << "Book Title" << setw(10) << "Genre" << setw(14) << "Year Publish" << setw(31) <<"Publisher" << endl;
    for (int i=0; i<count; i++){
        if (books[i] != nullptr)
        {
            cout << i + 1 << ") ";
            books[i]->display();
            cout << endl;
        }
    }
}
```

## SPECIAL METHODS

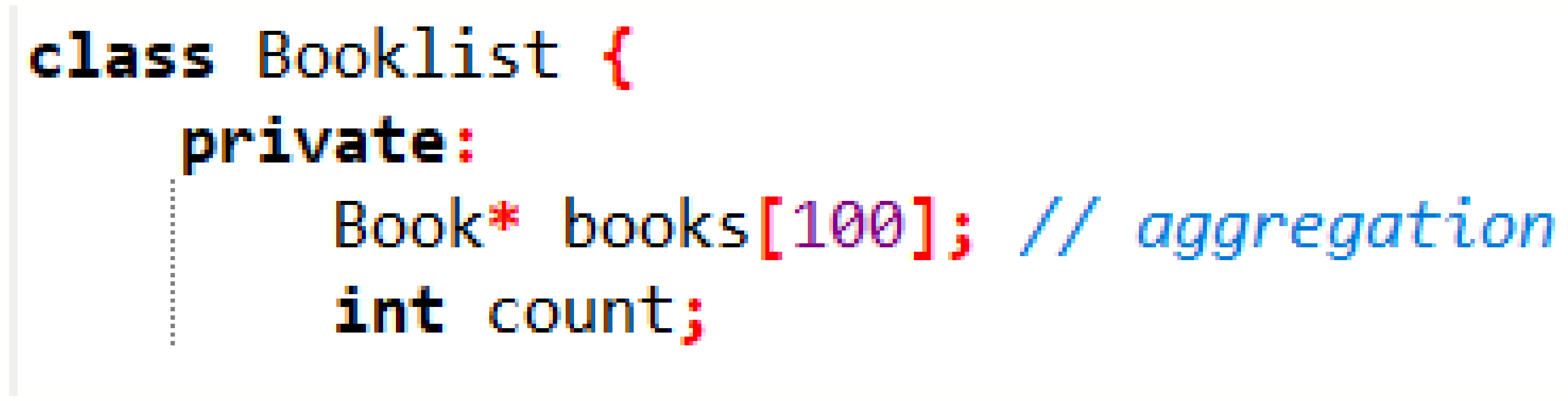
```
void removeBook(int index) {
    if (index >= 0 && index < count && books[index] != nullptr) {
        books[index] = nullptr;
        for (int i = index; i < count - 1; i++) {
            books[i] = books[i + 1];
        }
        books[count - 1] = nullptr;
        count--;
        cout << "Book removed successfully." << endl;
    } else {
        cout << "Invalid book index." << endl;
    }
}
```

## SPECIAL METHODS



```cpp
void saveBooklist() {
    string filename = "user_booklist.txt";
    ofstream file(filename);
    if (!file) {
        cerr << "Failed to open file for saving." << endl;
        return;
    }
    file << "-------------------------------------------------------------------------------------------------------------------\n";
    file << "\t\t\t\t\t       Personal Book List   \t\t\t\t\t" << endl;
    file << "-------------------------------------------------------------------------------------------------------------------\n";
    file << left << setw(15) << "Book Code" << setw(25) << "Book Title" << setw(20) << "Genre" << setw(20) << "Year Publish" << setw(31) << "Publisher" << endl;
    file << "-------------------------------------------------------------------------------------------------------------------\n";
    for (int i = 0; i < count; i++) {
        if (books[i] != nullptr) {
            Publisher* publisher = books[i]->getPublisher();
            file << i + 1 << ") " << left << setw(12) << books[i]->getBookCode()
                 << setw(25) << books[i]->getBookTitle() << setw(20) << books[i]->getGenre() << setw(20)
                 << books[i]->getYearPublish() << setw(31) << publisher->getPublisherName() + ", " + publisher->getCountry() << endl;
        }
    }
    file.close();
    cout << "Booklist saved to " << filename << "." << endl;
}
```

## AGGREGATION



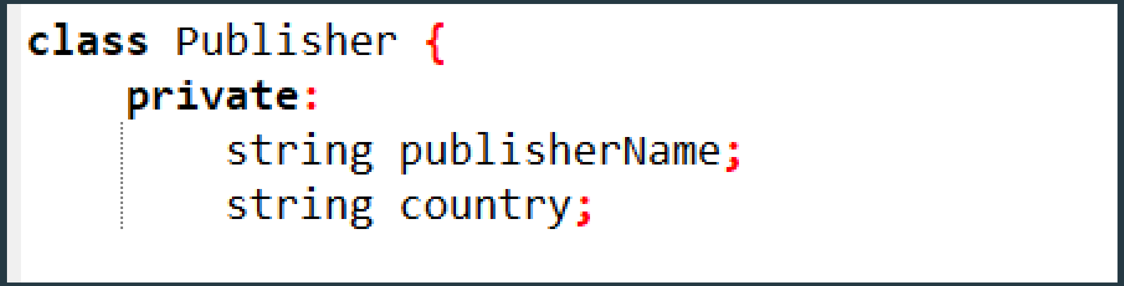
```
class Booklist {
    private:
        Book* books[100]; // aggregation
        int count;
```

# PUBLISHER CLASS

## Encapsulation :

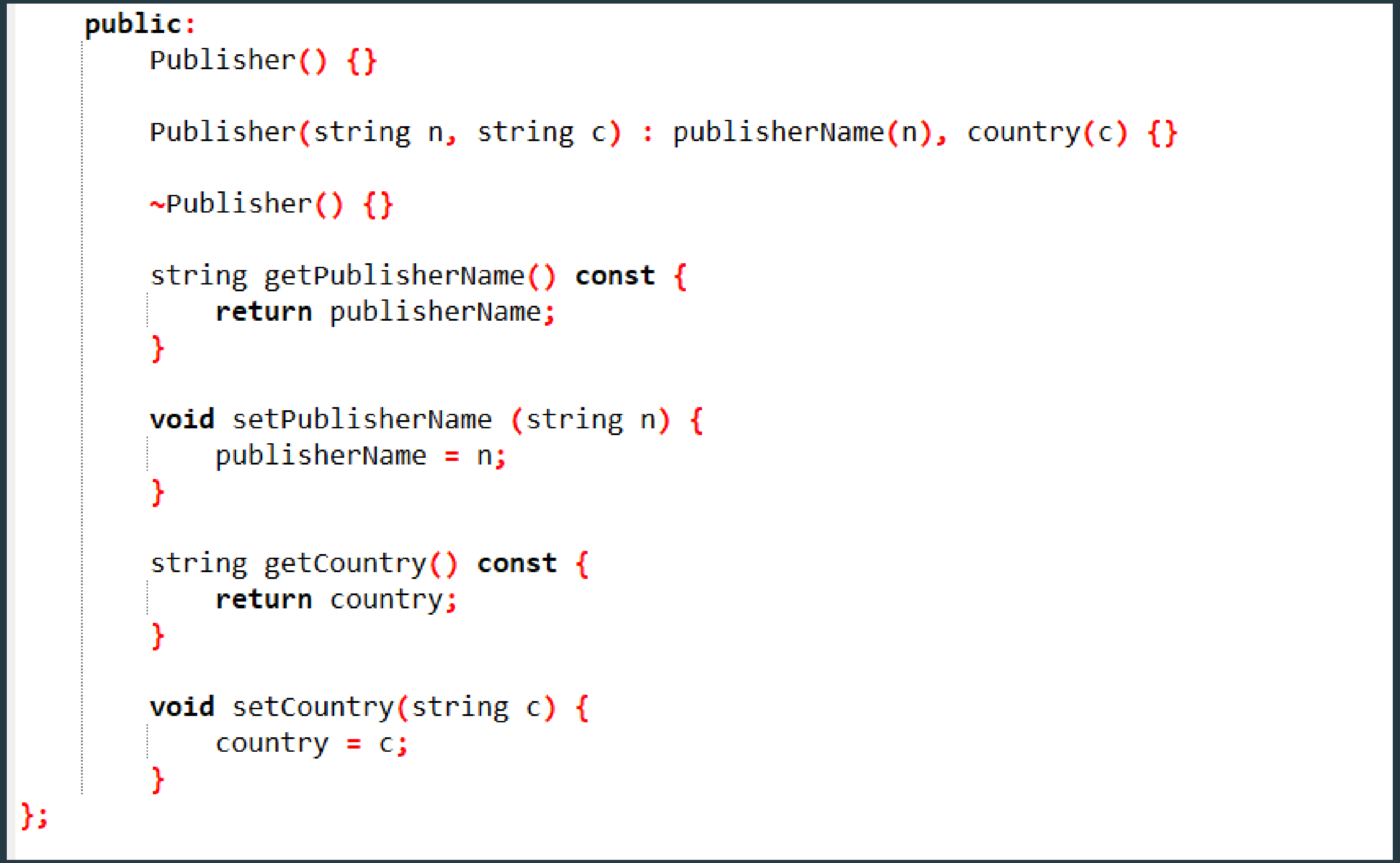
The Publisher class encapsulates the details of the publisher of the books

### Attributes



```cpp
class Publisher {
    private:
        string publisherName;
        string country;
```

### Methods



```cpp
    public:
        Publisher() {}

        Publisher(string n, string c) : publisherName(n), country(c) {}

        ~Publisher() {}

        string getPublisherName() const {
            return publisherName;
        }

        void setPublisherName (string n) {
            publisherName = n;
        }

        string getCountry() const {
            return country;
        }

        void setCountry(string c) {
            country = c;
        }
};
```

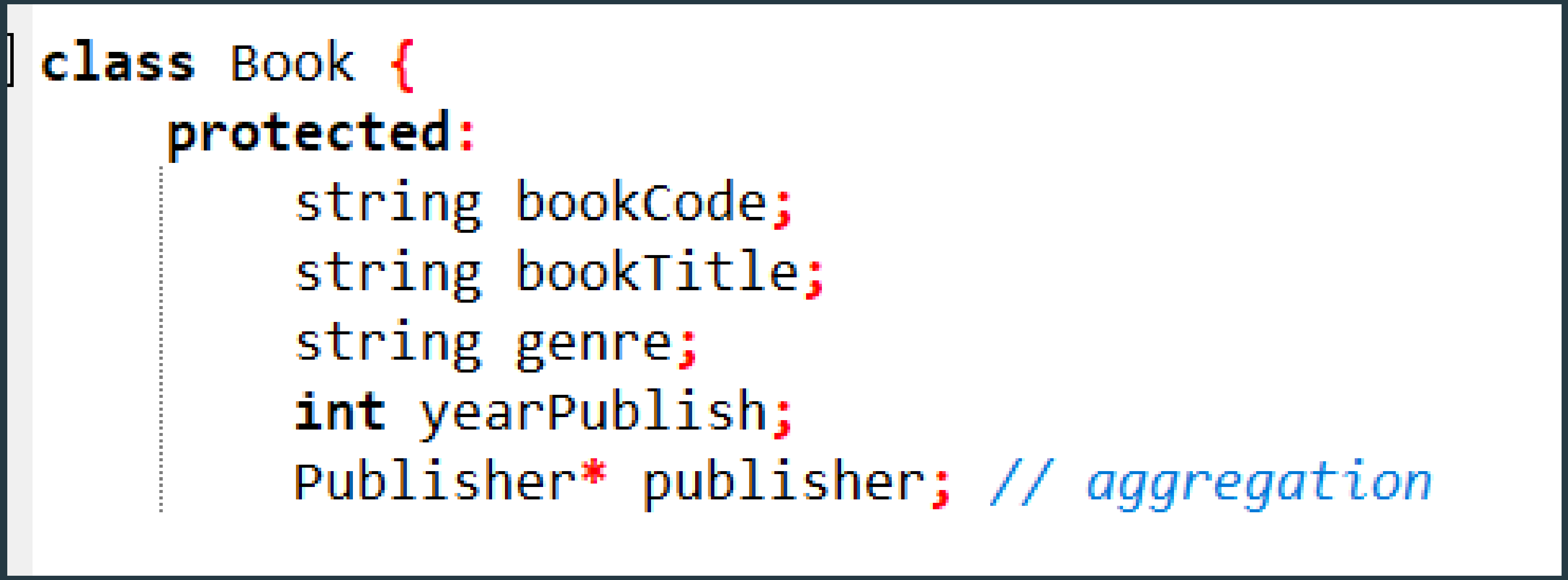
# BOOK CLASS

**Encapsulation :**

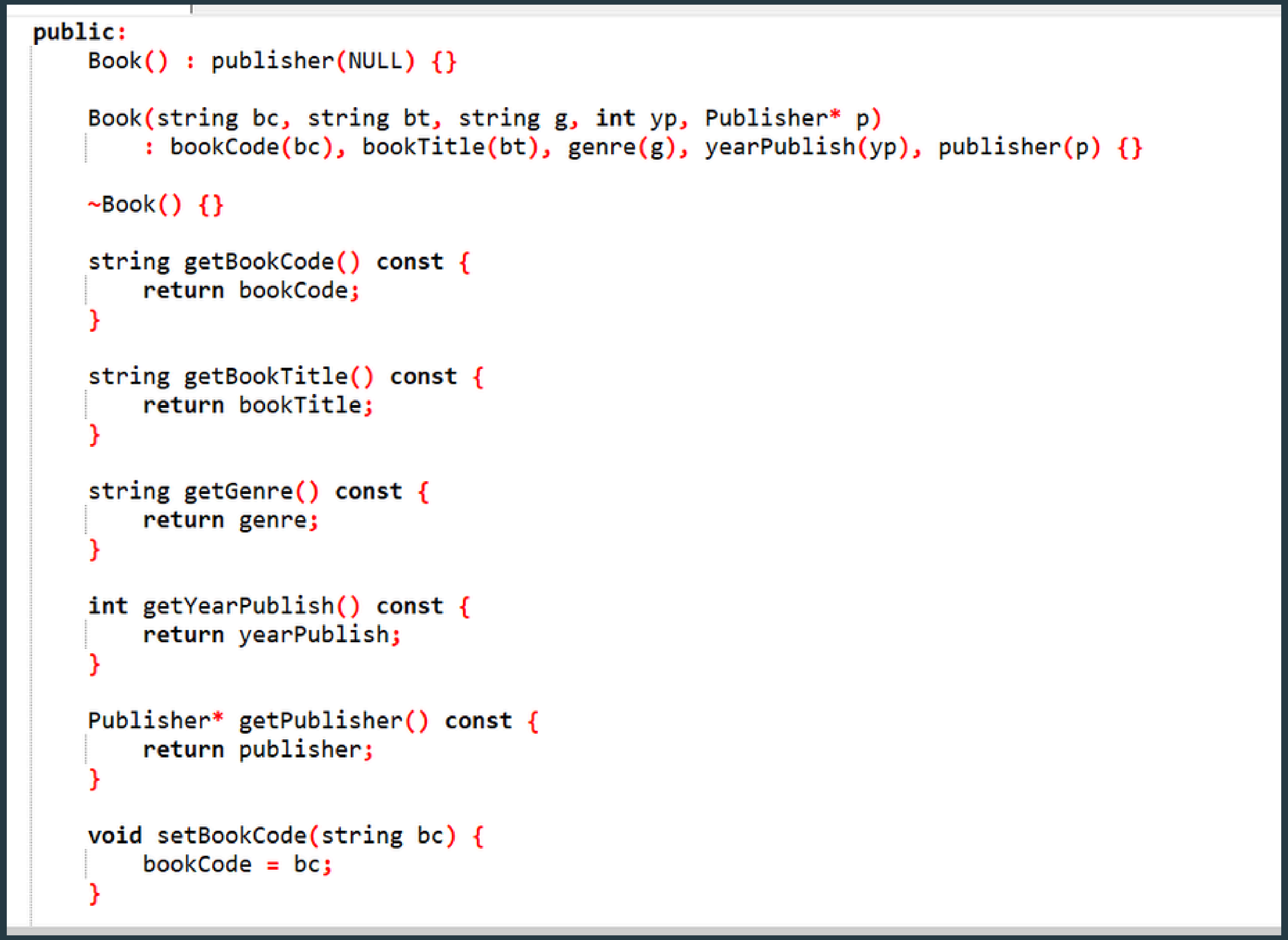
The Book class encapsulates the details of the books

**Attributes**

**Methods**



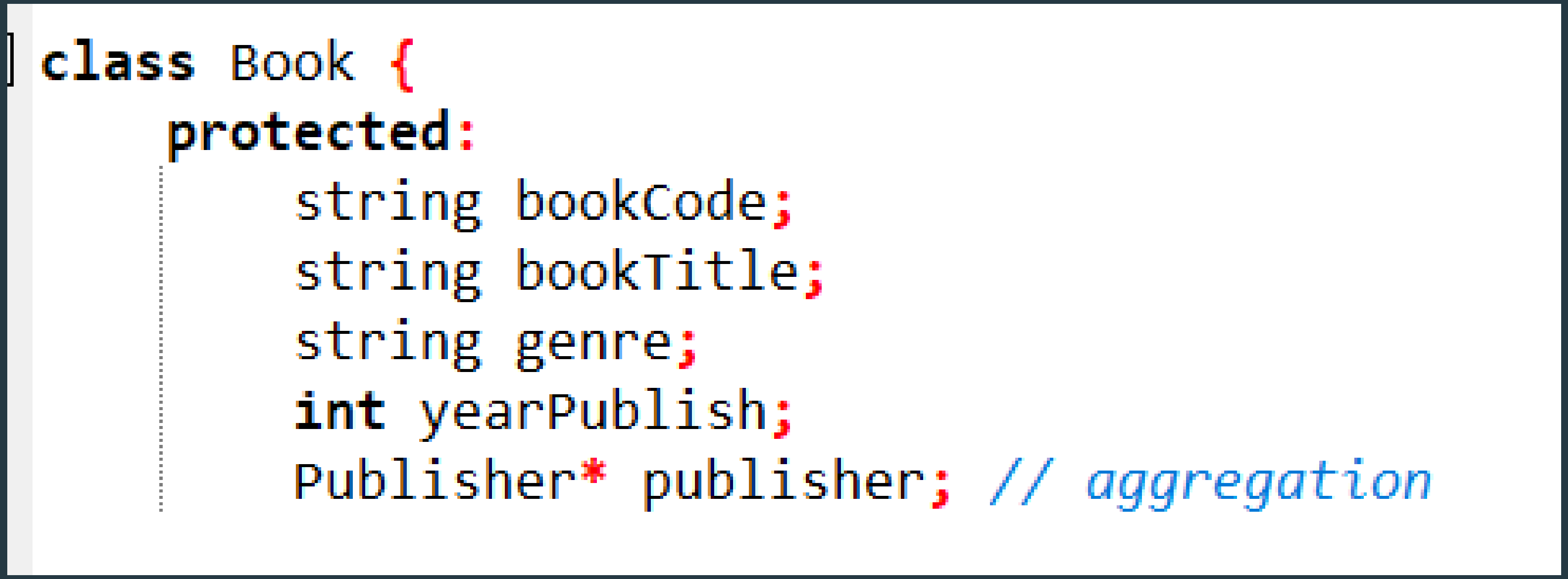
```
class Book {
    protected:
        string bookCode;
        string bookTitle;
        string genre;
        int yearPublish;
        Publisher* publisher; // aggregation
```



```
public:
    Book() : publisher(NULL) {}

    Book(string bc, string bt, string g, int yp, Publisher* p)
        : bookCode(bc), bookTitle(bt), genre(g), yearPublish(yp), publisher(p) {}

    ~Book() {}

    string getBookCode() const {
        return bookCode;
    }

    string getBookTitle() const {
        return bookTitle;
    }

    string getGenre() const {
        return genre;
    }

    int getYearPublish() const {
        return yearPublish;
    }

    Publisher* getPublisher() const {
        return publisher;
    }

    void setBookCode(string bc) {
        bookCode = bc;
    }
```

# BOOK CLASS

## Aggregation



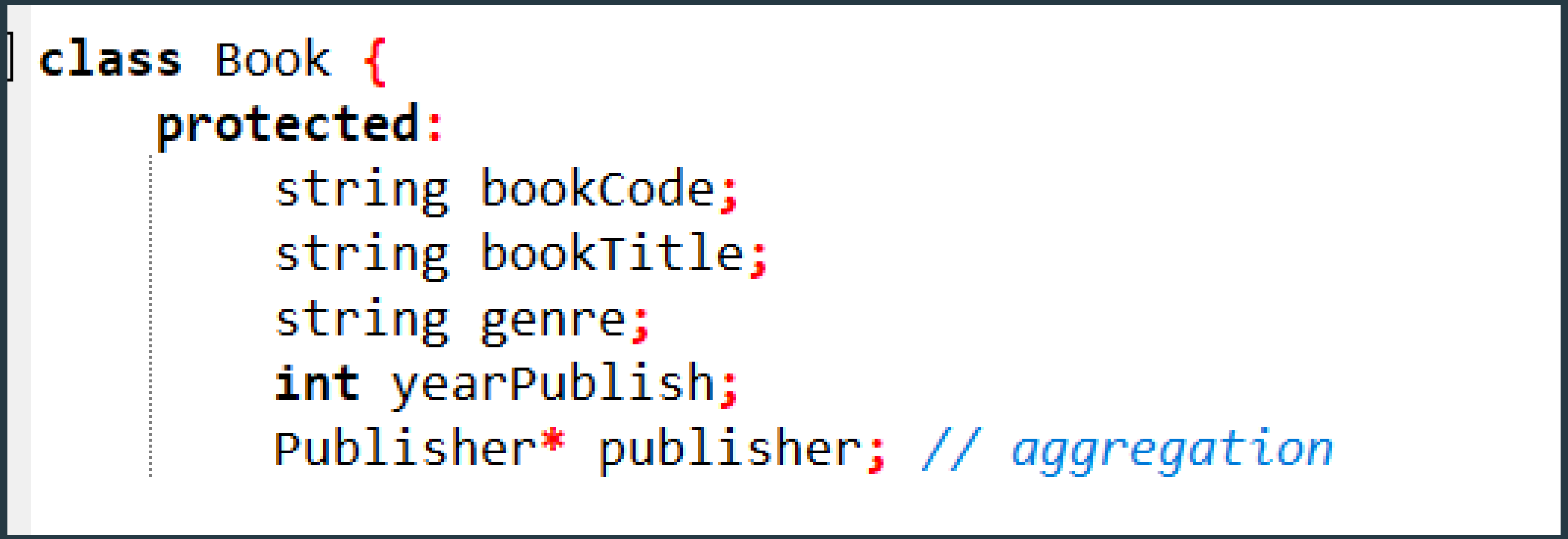
```
class Book {
    protected:
        string bookCode;
        string bookTitle;
        string genre;
        int yearPublish;
        Publisher* publisher; // aggregation
```

- The Book class holds a pointer to a Publisher object.
- The Publisher can exist independently. If the Book object is destroyed, the Publisher object will not be destroyed.

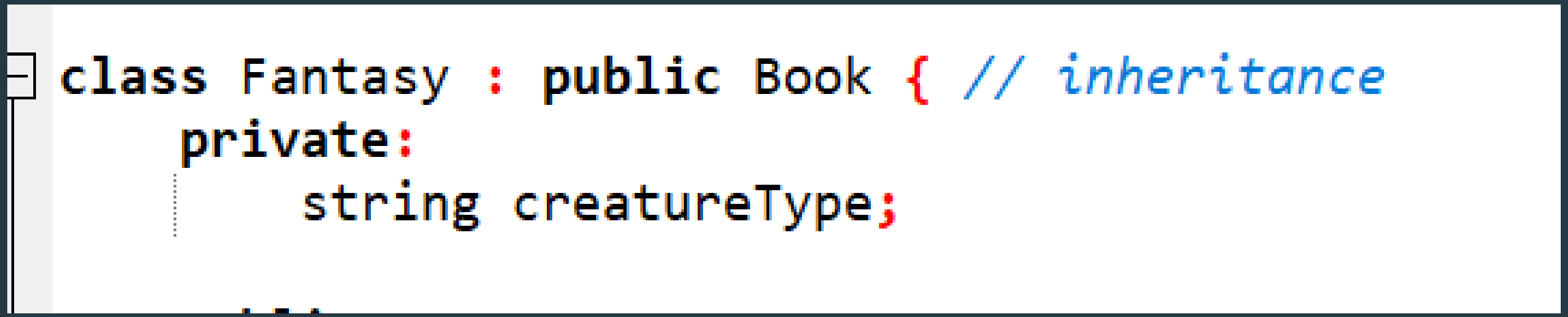
# BOOK CLASS

## Inheritance

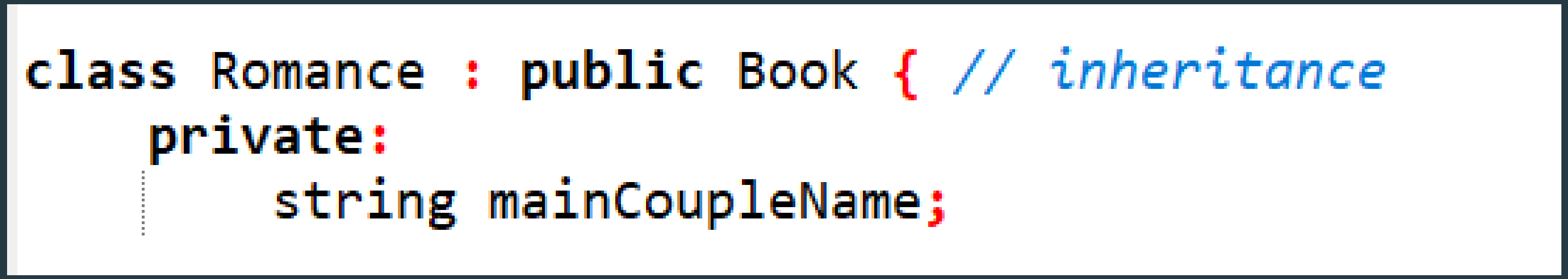
Book class is the base class of Romance, Fantasy and SciFi class



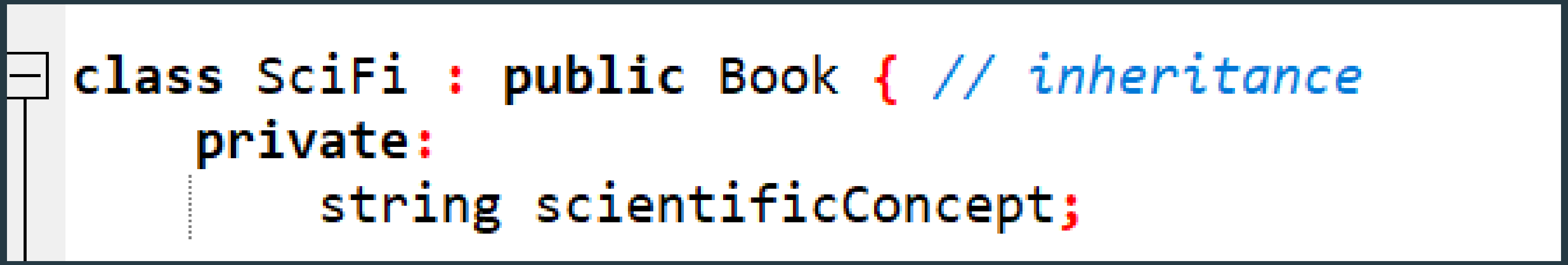
```
class Book {
    protected:
        string bookCode;
        string bookTitle;
        string genre;
        int yearPublish;
        Publisher* publisher; // aggregation
```



```
class Fantasy : public Book { // inheritance
    private:
        string creatureType;
```



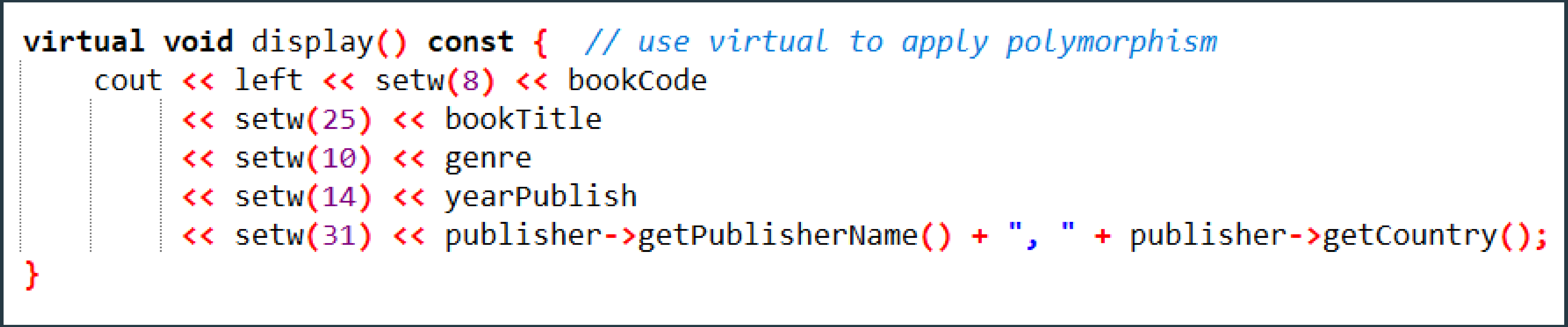
```
class Romance : public Book { // inheritance
    private:
        string mainCoupleName;
```



```
class SciFi : public Book { // inheritance
    private:
        string scientificConcept;
```

# BOOK CLASS

## Polymorphism

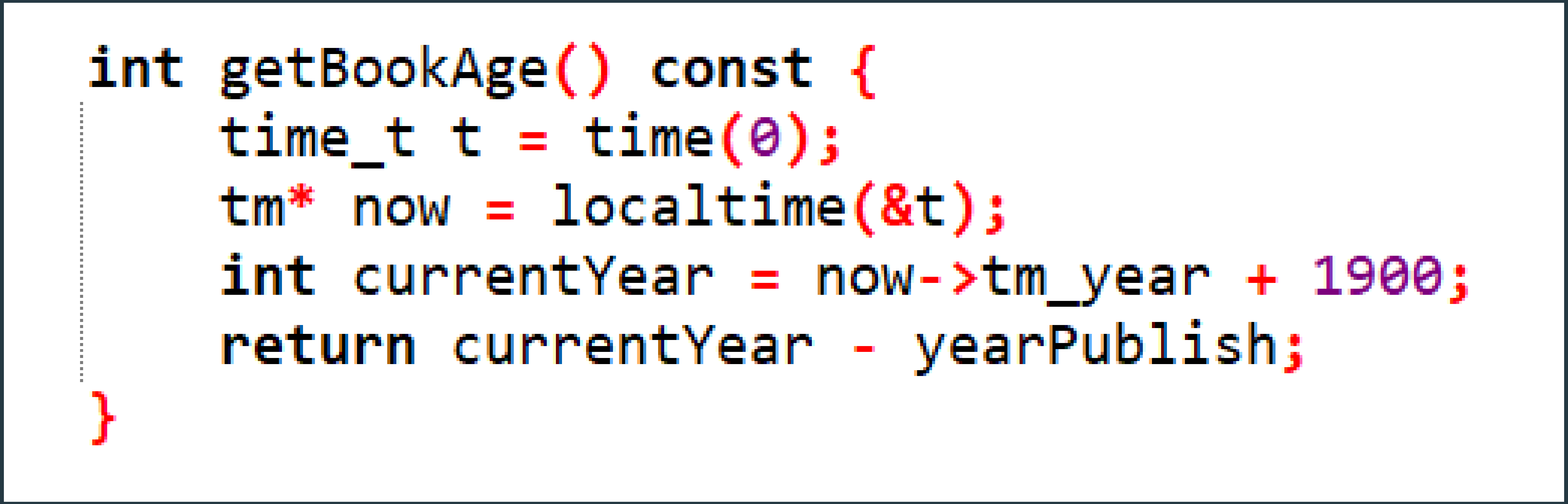


```
virtual void display() const {  // use virtual to apply polymorphism
    cout << left << setw(8) << bookCode
         << setw(25) << bookTitle
         << setw(10) << genre
         << setw(14) << yearPublish
         << setw(31) << publisher->getPublisherName() + ", " + publisher->getCountry();
}
```

- Virtual display method is declared in Book class to be overridden by derived classes.

# BOOK CLASS

## Special method

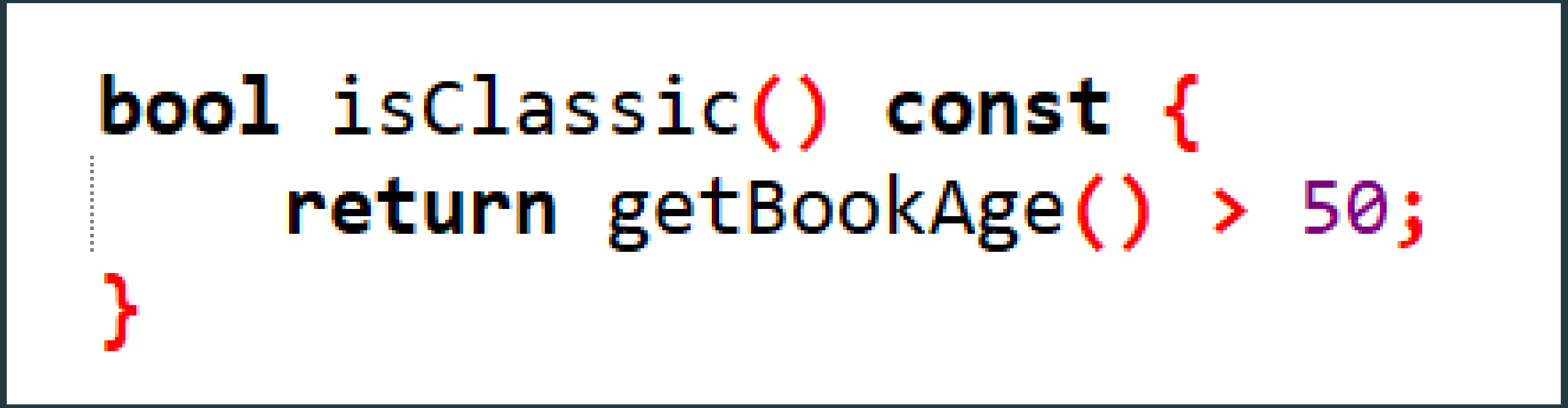


```cpp
int getBookAge() const {
    time_t t = time(0);
    tm* now = localtime(&t);
    int currentYear = now->tm_year + 1900;
    return currentYear - yearPublish;
}
```

- To calculate the age of the book

# BOOK CLASS

## Special method



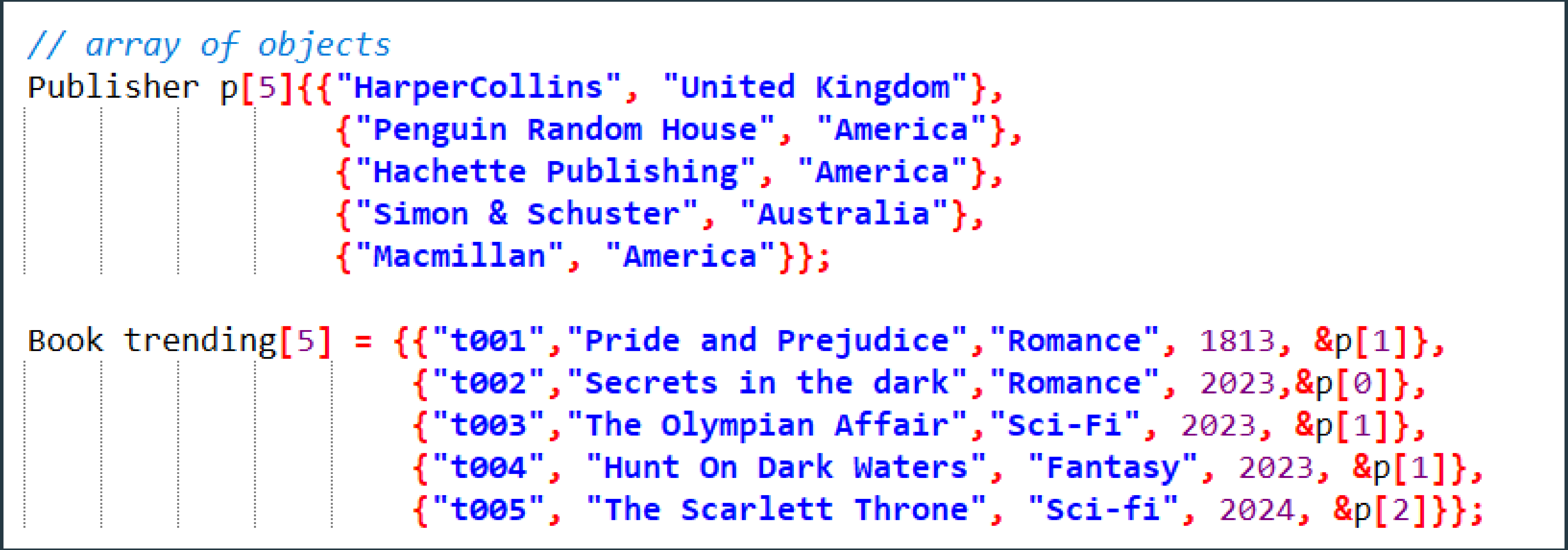
```
bool isClassic() const {
    return getBookAge() > 50;
}
```

- If the age of the book is more than 50, then it is considered as Classic book

# MAIN FUNCTION

**Array of Objects**

Used to manage collections of books and publishers.



```
// array of objects
Publisher p[5]{{"HarperCollins", "United Kingdom"},
               {"Penguin Random House", "America"},
               {"Hachette Publishing", "America"},
               {"Simon & Schuster", "Australia"},
               {"Macmillan", "America"}};

Book trending[5] = {{"t001","Pride and Prejudice","Romance", 1813, &p[1]},
                    {"t002","Secrets in the dark","Romance", 2023,&p[0]},
                    {"t003","The Olympian Affair","Sci-Fi", 2023, &p[1]},
                    {"t004", "Hunt On Dark Waters", "Fantasy", 2023, &p[1]},
                    {"t005", "The Scarlett Throne", "Sci-fi", 2024, &p[2]}};
```

## DEMOSTRATION

# THANK YOU FOR LISTENING