编程题目：图书馆借阅费用计算系统

题目描述

你需要在一个图书馆系统中实现借阅费用的计算。图书馆里有多本书，每本书都有一个唯一的ID。借阅每本书时可以选择不同的费用计算策略，例如，按天计费，或者会员按周付费。

你需要完成以下功能：

1. **借阅书籍**：借阅一本书，根据书籍ID和借阅策略记录借阅信息。
2. **归还书籍**：统一归还所有书籍，根据借阅的天数计算并返回借阅总费用。
3. **实现借阅计费策略**：
    1. **按日收费**：每日收取1块钱。
    2. **会员按周收费**：原始费用乘以相应会员的折扣；
        - 原始费用：每周收取5块钱，天数不足一周按一周计算，如：8天算作2周；
        - 会员折扣（包含三种会员）：`REGULAR`：0.95；`GOLD`：0.85，`PLATINUM`：0.75。

我们主要的测试过程如下：

1. 每次借出一本书籍，并指定其借阅计费策略，其中会员按周收费策略中会指定会员等级；
2. 归还所有书，所有书籍的借阅天数是一致的，结算总费用；
3. 验证总费用是否正确。
4. 重复这个过程。

框架代码说明

- `Book`类：表示一本书，包含ID和借阅计费策略。
- `BorrowStrategy`类：一个抽象基类，定义了计算借阅费用的接口。
- `ConcreteBorrowStrategy`类：`BorrowStrategy`的一个具体实现，实现具体的计费规则。
- `Library`类：图书馆管理系统，包含添加书籍、借阅书籍和归还书籍的功能。
- 你可以通过输入测试案例的名称来运行相应的测试。

具体要求

1. **实现`Book`类**：
    - 修改`calculate_fee`方法，使其根据借阅计费策略计算费用。
2. **实现`Library`类**：
    - `borrow_book`方法：接收两个参数：书籍id以及借阅策略；借阅一本书，指定其借阅计费策略。
    - `return_books`方法：归还所有书籍，根据借阅的天数和书籍的借阅策略计算总费用，接收一个参数：天数。
3. **实现`MemberBorrowStrategy`类**
    - 包含会员等级，string类型表示，只包括这些值："REGULAR"；"GOLD"； "PLATINUM"。
    - `calculate`方法：实现题目描述中的会员按周计费规则。

代码框架

下面是代码的框架，你需要补充具体的实现。

```
#include <iostream>
#include <memory>
#include <string>
#include <cassert>
#include <unordered_map>
#include <functional>

class BorrowStrategy {
public:
    virtual ~BorrowStrategy() = default;

    virtual double calculate(int days) const = 0;
};

class DailyBorrowStrategy : public BorrowStrategy {
public:
    DailyBorrowStrategy() {
    }

    double calculate(int days) const override {
        return days * 1.0;
    }
};

// todo: complete MemberBorrowStrategy
class MemberBorrowStrategy : public BorrowStrategy {
public:
    MemberBorrowStrategy(std::string level)
        : memberLevel(level) {
    }

    double calculate(int days) const override {
    }

private:
    std::string memberLevel;
};


class Book {
public:
    Book(int id) : id(id) {
    }

    double calculate_fee() const {
        // todo: add codes here
        return 0.0;
    }

    int get_id() const { return id; }

    void setBorrowStrategy(std::shared_ptr<BorrowStrategy> strategy) {
        borrow_strategy = strategy;
```

```
    }

private:
    int id;
    std::shared_ptr<BorrowStrategy> borrow_strategy;
};

class Library {
public:
    using BookPtr = std::shared_ptr<Book>;

    void add_book(const BookPtr &book) {
        books[book->get_id()] = book;
    }

    void borrow_book(int id, const std::shared_ptr<BorrowStrategy> &strategy) {
        // todo: add codes here
    }

    double return_books(int days) {
        // todo: add codes here
        return 0.0;
    }

private:
    std::unordered_map<int, BookPtr> books;
};

// === TEST_CASES ===
void TEST_1();

void TEST_2();

void TEST_3();

void TEST_4();

void TEST_5();

void TEST_6();

void TEST_7();


#define REGISTER_TEST_CASE(name) {#name, name}

int main() {
    std::unordered_map<std::string, std::function<void()> >
            test_functions_by_name = {
                REGISTER_TEST_CASE(TEST_1), REGISTER_TEST_CASE(TEST_2),
                REGISTER_TEST_CASE(TEST_3), REGISTER_TEST_CASE(TEST_4),
                REGISTER_TEST_CASE(TEST_5), REGISTER_TEST_CASE(TEST_6),
                REGISTER_TEST_CASE(TEST_7),
            };

    std::string test_case_name;
    std::cin >> test_case_name;
    auto it = test_functions_by_name.find(test_case_name);
```

```
    assert(it != test_functions_by_name.end());
    auto fn = it->second;
    fn();
    return 0;
}

void TEST_1() {
    Library library;

    library.add_book(std::make_shared<Book>(1));
    library.add_book(std::make_shared<Book>(2));

    library.borrow_book(1, std::make_shared<DailyBorrowStrategy>());

    assert(library.return_books(5) == 5.0);
}

void TEST_2() {
    Library library;

    library.add_book(std::make_shared<Book>(1));
    library.add_book(std::make_shared<Book>(2));

    library.borrow_book(2, std::make_shared<MemberBorrowStrategy>("GOLD"));

    assert(library.return_books(8) == 8.5);
}

void TEST_3() {
    Library library;

    library.add_book(std::make_shared<Book>(1));

    assert(library.return_books(0) == 0.0);
}

void TEST_4() {
    Library library;

    library.add_book(std::make_shared<Book>(1));
    library.add_book(std::make_shared<Book>(2));

    library.borrow_book(1, std::make_shared<DailyBorrowStrategy>());
    library.borrow_book(2, std::make_shared<MemberBorrowStrategy>("PLATINUM"));

    assert(library.return_books(10) == 17.5);
}

void TEST_5() {
    Library library;

    library.add_book(std::make_shared<Book>(1));
    library.add_book(std::make_shared<Book>(2));

    library.borrow_book(1, std::make_shared<DailyBorrowStrategy>());

    assert(library.return_books(5) == 5.0);
```

```
    library.borrow_book(2, std::make_shared<MemberBorrowStrategy>("REGULAR"));

    assert(library.return_books(7) == 4.75);
}

void TEST_6() {
    Library library;

    library.add_book(std::make_shared<Book>(1));
    library.add_book(std::make_shared<Book>(2));

    library.borrow_book(2, std::make_shared<MemberBorrowStrategy>("PLATINUM"));

    assert(library.return_books(14) == 7.5);
}

void TEST_7() {
    Library library;

    library.add_book(std::make_shared<Book>(1));
    library.add_book(std::make_shared<Book>(2));

    library.borrow_book(1, std::make_shared<DailyBorrowStrategy>());

    assert(library.return_books(5) == 5.0);

    assert(library.return_books(10000) == 0.0);
}
```