TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM

ĐỀ THI KẾT THÚC HỌC PHẦN
Học kỳ I – Năm học 2021-2022

MÃ LƯU TRỮ (*do phòng KT-ĐBCL ghi*)

Tên học phần:	**Kiến trúc phần mềm**	**Mã HP:**	**CSC13106**
Thời gian làm bài:	**120 phút**	Ngày thi:	**16/01/2022**

Ghi chú: *Sinh viên* [☐ *được phép* / ☑ *không được phép*] *sử dụng tài liệu khi làm bài.*

Lưu ý: **Sinh viên có thể sử dụng bất kỳ ngôn ngữ lập trình phổ biến nào để trả lời các câu hỏi.**

PHẦN I – HỆ THỐNG ỨNG DỤNG PHI TẬP TRUNG (DECENTRALIZED APPS)

Decentralized Application (DApp) là một khuynh hướng mới của phát triển hệ thống và ứng dụng. Trong các ứng dụng truyền thống, bao gồm cả ứng dụng tập trung (**Centralized Application**) hay ứng dụng phân tán (**Distributed Application**), mỗi thao tác xử lý chỉ được thực hiện 1 lần tại 1 máy tính hay 1 cụm máy tính. Với ứng dụng DApp, mọi dữ liệu và xử lý được ***lưu trữ*** và ***thực thi trên tất cả các máy tính/thiết bị tính toán*** (gọi chung là các ***node***) tham gia vào hệ thống, hướng đến hạn chế việc một tổ thức/thực thể nào có thể xóa/sửa nội dung hay kết quả xử lý.

Nội dung dữ liệu hay kết quả tính toán chỉ được ***công nhận là hợp lệ*** khi ***đa số các node trong hệ thống đều lưu trữ hay tính ra cùng kết quả này***. Bất kỳ khi nào giá trị dữ liệu được sửa đổi theo 1 thao tác hợp lệ, giá trị mới của dữ liệu sẽ được đồng bộ hóa trên tất cả các node. Điều này dường như không hiệu quả khi phải tạo ra bản sao dữ liệu cũng như xử lý cùng 1 tính toán trên mọi node của hệ thống nhưng giải pháp này cho phép thay đổi cách tiếp cận trong phát triển hệ thống từ hướng tập trung sang hướng ***phi tập trung*** với sự tham gia của tất cả các node thành viên vào quá trình tính toán và duy trì tính đúng đắn của dữ liệu.

Để hạn chế việc lạm dụng/tấn công làm quá tải hệ thống, các thao tác tính toán hay ghi nhận dữ liệu thường đòi hỏi người gọi thực thi thao tác cần ***sử dụng*** một lượng ***token*** hay một khoản ***chi phí ảo*** (thường tương ứng với chi phí tính toán/lưu trữ). Khái niệm này được hiện thực hóa thành `ether` (trong Ethereum), `bitcoin` (trong Bitcoin), hay `energy` (trong đề bài này).

Nhóm nghiên cứu *Futuristic Integration Technology* (**FIT**) để phát triển một platform đơn giản cho việc xây dựng và vận hành các ứng dụng DApp. Giải pháp này được xây dựng kế thừa một số ý tưởng, khái niệm của các giải pháp về DApp hiện có, nhưng cũng có một số đặc tính và cách tiếp cận riêng.

Trong đề bài này, anh/chị sẽ làm quen 1 số thành phần trong platform được xây dựng cho ứng dụng DApp. Cần lưu ý là các vấn đề trong đề bài được đơn giản hóa để phù hợp với ngữ cảnh của đề thi.

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM

ĐỀ THI KẾT THÚC HỌC PHẦN
Học kỳ I – Năm học 2021-2022

MÃ LƯU TRỮ (do phòng KT-ĐBCL ghi)

Lớp MEntity

Lớp ảo `MEntity` là lớp đối tượng tổng quát cho mọi lớp đối tượng trong hệ thống.

```
public abstract class MEntity {
    // Tự điển tất cả kiểu dữ liệu trên hệ thống
    protected static Dictionary<string, MEntity> entities =
                new Dictionary<string, MEntity>();
    protected static int Counter = 0;   // Tổng số đối tượng MEntity
    public    string ID;                // Định danh duy nhất của đối tượng MEntity
    public    string ClassName;         // Tên của lớp đối tượng, sẽ thay đổi tùy theo các lớp kế thừa.
    public MEntity() {…}
    // Tìm và trả về đối tượng có định danh là ID và loại đối tượng tên là ClassName. Nếu không có, trả về null
    public static MEntity FindByID(string ID, string strClassName) {…}
}
```

Câu 1. Khởi tạo mặc định cho mọi đối tượng `MEntity` **(0,5 điểm)**

Mỗi đối tượng `MEntity` có `ID` ***dạng chuỗi phân biệt***, được xác định bằng cách gọi phương thức tĩnh `SHA3_512.Hash(Counter++)` (đã được cài đặt sẵn). Khi đối tượng `MEntity` được tạo ra, đối tượng này luôn ***tự đăng ký*** vào tự điển `MEntity.entities` với khóa dạng chuỗi là `ID` của mình.

Hãy ***cài đặt*** `constructor` của lớp `MEntity`.

Câu 2. Tìm và trả về đối tượng `MEntity` theo định danh ID và loại đối tượng cụ thể **(0,5 điểm)**

Hãy ***cài đặt*** hàm `FindByID()` cho lớp `MEntity` để tìm và trả về đối tượng có tên lớp là `strClassName` và định danh là `ID`. Nếu không có, trả về `null`.

🕭 ***Hướng dẫn:*** Cần kiểm tra đối tượng `MEntity` có định danh là `ID` được tìm thấy trong tự điển có thuộc tính `ClassName` trùng với `strClassName` hay không.

Lớp MAccount

Mỗi tài khoản (`MAccount`, kế thừa từ `MEntity`) có năng lượng (`Energy`) để thực hiện các thao tác trên hệ thống. Năng lượng trong tài khoản bị trừ đi khi thực hiện thao tác trên hệ thống (như tính toán, tạo kiểu dữ liệu, tạo hay cập nhật đối tượng dữ liệu…) và có thể được bổ sung/thưởng thêm.

```
public class MAccount: MEntity {
    public float Energy;  // năng lượng hiện tại của tài khoản để hoạt động trên hệ thống
    // Kiểm tra tài khoản có đủ năng lượng để thực thi 1 thao tác trên hệ thống
    public bool EnoughEnergy(float amount)  { return Energy >= amount; }
    // Sử dụng năng lượng để thực thi 1 thao tác trên hệ thống
    public void ConsumeEnergy(float amount) { Energy -= amount; }
    // Nhận thêm năng lượng
    public void SupplyEnergy(float amount)  { Energy += amount; }
}
```

Lớp MType

Để có thể viết ra ứng dụng DApp, việc trước tiên là cần định nghĩa và đăng ký các kiểu dữ liệu mới. Lớp đối tượng `MType` cho phép mô tả 1 kiểu dữ liệu mới, bao gồm tên kiểu dữ liệu (`Name`, dạng chuỗi) và danh sách tên các thuộc tính (`AttributeList`, mảng các chuỗi).

```
public class MType : MEntity {
    static private float energyCreate = 10;    // năng lượng cần tiêu thụ khi đăng ký 1 kiểu dữ liệu
    public string Name;    // Tên kiểu dữ liệu
    public List<string> AttributeList = new List<string>(); // Danh sách tên các thuộc tính
    static public  MType GetType(string ID) { // Tìm và ép kiểu MType cho đối tượng có ID cho trước
        MEntity res = MEntity.FindByID(ID, "MType");
        if (res != null) return (MType)res;
        return null;
    }
}
```

Câu 3. Khởi tạo và đăng ký kiểu dữ liệu mới **(0,5 điểm)**

Mỗi khi tạo ra 1 kiểu dữ liệu mới, tài khoản (`MAccount`) tạo ra kiểu dữ liệu này cần phải tiêu thụ `MType.energyCreate` đơn vị năng lượng.

Phương thức tĩnh `RegisterNewType()` trong lớp `MType` được dùng để tạo ra và đăng ký 1 kiểu dữ liệu mới:

```
static public string RegisterNewType( string AccountID,
                                      string name, List<string> attributeList)
```

Hãy ***đề xuất giải pháp*** để đảm bảo **<u>không</u>** thể tạo ra đối tượng thuộc lớp `MType` bên ngoài lớp `MType` mà phải thông qua hàm `MType.RegisterNewType()`. Sau đó, ***cài đặt*** phương thức tĩnh `MType.RegisterNewType()`để tạo ra 1 đối tượng `MType` mới có tên kiểu dữ liệu là `name` và danh sách tên các thuộc tính là `attributeList`. Định danh của tài khoản thực hiện đăng ký kiểu dữ liệu là `AccountID`. Kết quả trả về của hàm là `ID` của đối tượng `MType` được tạo ra.

🖃 ***Hướng dẫn:*** Trong hàm `RegisterNewType()` cần lưu ý:

- Tìm tài khoản `MAccount` tương ứng với `AccountID`
- Nếu tài khoản tồn tại và còn đủ năng lượng (tối thiểu là `MType.energyCreate`) thì:
  - Trừ lượng năng lượng `MType.energyCreate` vào tài khoản tương ứng
  - Tạo ra đối tượng `MType` với tên kiểu dữ liệu là `name` và danh sách tên các thuộc tính là `attributeList`
  - Trả về `ID` của đối tượng `MType` vừa được tạo ra.
- Trả về chuỗi rỗng nếu việc thực hiện không thành công.

Lớp MObject

Lớp đối tượng `MObject` cho phép lưu trữ thông tin của các đối tượng dữ liệu khác nhau. `energyCreate` và `energyUpdate` là năng lượng tiêu thụ khi tạo ra hay cập nhật 1 đối tượng dữ liệu. Tự điển `Attributes` chứa tất cả các thuộc tính trong đối tượng `MObject`.

```
public class MObject : MEntity {
    static private float energyCreate = 2;      // năng lượng cần khi tạo ra 1 đối tượng dữ liệu
    static private float energyUpdate = 1;      // năng lượng cần khi cập nhật 1 đối tượng dữ liệu
    public string TypeID;                       // ID của kiểu dữ liệu của đối tượng dữ liệu
    // Tự điển  các thuộc tính. Giá trị của mỗi thuộc tính được lưu ở kiểu chuỗi.
    public Dictionary<string, string> Attributes = new Dictionary<string, string>();
    // Constructor để tạo ra đối tượng dữ liệu tương ứng với kiểu dữ liệu có định danh là TypeID
    protected MObject(string TypeID){…}
    // Tài khoản có định danh là AccountID tạo ra đối tượng với kiểu dữ liệu có định danh là TypeID
    static public string CreateNewObject(string AccountID,string TypeID){…}
    // Tìm và ép kiểu MObject cho đối tượng có ID cho trước
    static public MObject GetObject(string ID){…}
    // Lấy giá trị hoặc cập nhật giá trị thuộc tính có tên thuộc tính là strAttributeName
    public string this[string strAttributeName]{…}
}
```

Phương thức `CreateNewObject()` được cài đặt tương tự phương thức `RegisterNewType()` của lớp `MType` (xem ở câu 3). Trong hàm này cần kiểm tra xem tài khoản tương ứng với AccountID có còn đủ năng lượng (tối thiểu là `MObject.energyCreate`) trước khi đối tượng thật sự được tạo lập, đồng thời sẽ trừ `MObject.energyCreate` vào tài khoản tương ưng sau khi tạo xong đối tượng. Phương thức `GetObject()` được cài đặt tương tự phương thức `GetType()` của lớp `MType`.

Câu 4. Khởi tạo đối tượng `MObject` tương ứng với kiểu dữ liệu có định danh `TypeID` cho trước(0,5 điểm)

Tương tự constructor của lớp `MType`, hãy khai báo và cài đặt constructor cho lớp `MObject` để khởi tạo đối tượng tương ứng với kiểu dữ liệu có `TypeID` cho trước.

🖃 ***Hướng dẫn:*** Sử dụng phương thức tĩnh `MType.GetType()` để lấy được đối tượng `type` thuộc lớp `MType` tương ứng với `TypeID` cho trước. Sau đó, khởi tạo tự điển `Attributes` chứa đầy đủ các thuộc tính đã được khai báo trong danh sách tên các thuộc tính `type.AttributeList`.

Câu 5. Lấy và cập nhật giá trị thuộc tính bất kỳ trong đối tượng MObject (0,5 điểm)

Khai báo và cài đặt phương thức cho phép lấy giá trị và cập nhật giá trị thuộc tính có tên là `strAttributeName` trong đối tượng thuộc lớp `MObject`.

🖃 ***Hướng dẫn:***

- Khi lấy giá trị thuộc tính, nếu không tồn tại thuộc tính có tên là `strAttributeName` thì trả về chuỗi rỗng.
- Khi cập nhật giá trị thuộc tính: Nếu tồn tại thuộc tính có tên `strAttributeName` thì chỉ cập nhật thuộc tính nếu giá trị mới khác giá trị hiện có; nếu chưa tồn tại thuộc tính có tên `strAttributeName` thì bổ sung thuộc tính mới vào tự điển thuộc tính của đối tượng, sau đó gán giá trị vào.

Lớp `MTransaction`

Lớp đối tượng `MTransaction` cho phép định nghĩa 1 giao tác (transaction) để xử lý. Giao tác có thể đọc dữ liệu hiện có để tính toán và xử lý, tạo ra dữ liệu mới cũng như gọi các giao tác khác. `energyCreate` và `energyInvoke` là năng lượng tiêu thụ khi đăng ký giao tác và thực thi giao tác.

```
public abstract class MTransaction: MEntity {
    static private float energyCreate = 5;     // năng lượng cần khi đăng ký 1 giao tác
    static private float energyInvoke = 1;     // năng lượng cần khi thực thi 1 giao tác
    // Tìm và trả về giao tác MTransaction trong tự điển có định danh là ID. Nếu không có, trả về null
    static public MTransaction GetTransaction(string TransactionID){…}
    // Tài khoản có định danh là AccountID đăng ký 1 giao tác Transaction. Kết quả trả về là ID của Transaction
    static public string RegisterTransaction(string AccountID, MTransaction Transaction){…}
    // Tài khoản có định danh là AccountID thực thi giao tác.
    // Mảng input chứa danh sách các tham số đầu vào.
    public MObject InvokeTransaction(string AccountID, string[] input){…}
    // Phương thức nội bộ để xử lý giao tác. Tài khoản thực thi giao tác có định danh là AccountID
    // Mảng input chứa danh sách các tham số đầu vào.
    protected abstract MObject Execute(string AccountID, string[] input);
}
```

Câu 6. Hàm thực thi giao tác trong lớp `MTransaction` **(0,5 điểm)**

Hãy cài đặt hàm `InvokeTransaction` của lớp `MTransaction` theo khai báo sau:

```
    public MObject InvokeTransaction(string AccountID, string[] input){…}
```

Hàm `InvokeTransaction` được cài đặt theo mẫu `Template Method`:

- Sử dụng hàm `MEntity.FindByID()` hoặc `MAccount.GetAccount()` để tìm tài khoản `account` tương ứng với định danh `AccountID`.
- Nếu tài khoản `account` có còn đủ năng lượng tối thiểu là `MTransaction.energyCreate`:
  - Trừ lượng năng lượng `MTransaction.energyCreate` vào tài khoản `account`.
  - Gọi thực thi phương thức virtual/override `Execute()` và nhận về đối tượng `result` kiểu `MObject`
  - Trả đối tượng `result` ra ngoài
- Nếu hàm thực hiện không thành công: trả về `null`.

Câu 7. Một giao tác đơn giản **(0,5 điểm)**

Giao tác `DangKyHocPhan` kế thừa từ `MTransaction` cung cấp chức năng đăng ký học phần. Nhờ dùng mẫu Template Method trong phương thức `MTransaction.InvokeTransaction()`, chỉ cần cài đặt lại phương thức `Execute()` trong lớp `DangKyHocPhan` để xử lý nghiệp vụ cho việc đăng ký học phần.

Dựa vào những nội dung về kiến trúc phần mềm đã được mô tả từ đầu bài đến giờ và kiến thức thực tế, hãy giải thích rõ ý nghĩa của từng dòng lệnh (từ dòng 3 đến 25) trong mã nguồn của lớp `DangKyHocPhan`.

Dòng	Mã nguồn
1	`public class DangKyHocPhan : MTransaction {`
2	`  protected override MObject Execute(string AccountID, string[] input) {`
3	`      string  TypeID_DKHP = "0x20180105afebcc817123fa82bc";`
4	`      string  SinhVienID = input[0];`
5	`      string  MonHocID = input[1];`
6	`      int     sttLop = int.Parse(input[2]);`
7	`      int     HocKy = int.Parse(input[3]);`
8	`      string  NamHoc = input[4];`
9	`      MObject sinhvien = MObject.GetObject(SinhVienID);`
10	`      if (sinhvien == null) return null;`
11	`      MObject monhoc = MObject.GetObject(MonHocID);`
12	`      if (monhoc == null) return null;`
13	`      int SiSoToiDa = int.Parse(monhoc["SiSoToiDa"]);`
14	`      int SiSoHienTai = int.Parse(monhoc["SiSoHienTai"]);`
15	`      if ( SiSoHienTai < SiSoToiDa ) {`
16	`          monhoc["SiSoHienTai"] = (SiSoHienTai + 1).ToString();`
17	`          string ObjectID_DKHP = MObject.CreateNewObject(AccountID,TypeID_DKHP);`
18	`          MObject dkhp = MObject.GetObject(ObjectID_DKHP);`
19	`          dkhp["MaSV"] = sinhvien["MaSV"];`
20	`          dkhp["MaMH"] = monhoc["MaMH"];`
21	`          dkhp["STTLop"] = sttLop.ToString();`
22	`          dkhp["HocKy"] = NamHoc;`
23	`          return dkhp;`
24	`      }`
25	`      return null;`
26	`  }`
27	`}`

Lớp DecentralizedAppEngine

Đến lúc này, anh/chị đã có thể viết 1 ứng dụng đơn giản (dưới dạng 1 giao tác) để xử lý trên hệ thống. Tuy nhiên, cần lưu ý là khi gọi thực hiện 1 giao tác, giao tác này sẽ được xử lý độc lập trên tất cả mọi node của hệ thống, từ đó thu thập kết quả từ từng node. Kết quả của giao tác chỉ được công nhận và sử dụng nếu được sự đồng thuận của hầu hết các node.

```
public class DecentralizedAppEngine {
    // Tự điển toàn bộ các node xử lý và lưu trữ trong hệ thống
    private static Dictionary<string, MNode> nodes = new Dictionary<string, MNode>();
    // Yêu cầu xử lý giao tác có ID là TransactionID, tài khoản thực hiện có ID là AccountID, các tham số là input
    // Giao tác được xử lý trên toàn bộ các node, từ đó chọn kết quả thống nhất và trả về định danh kết quả
    public static string ProcesMTransaction(string AccountID, string TransactionID,
                                             string[] input) {…}
    // Tìm đối tượng dữ liệu được thống nhất cao nhất trong danh sách candidates
    private static MObject FindMostReliableResult(List<MObject> candidates) {…}
    // Yêu cầu cập nhật đối tượng dữ liệu có ID là ObjectID trên toàn bộ các node
    public static void UpdateObject(string ObjectID){…}
}
```

Câu 8. Tự động thông báo đồng bộ hóa đối tượng dữ liệu **(0,5 điểm)**

Khi 1 đối tượng dữ liệu được tạo ra hay thay đổi giá trị, cần đảm bảo đồng bộ hóa (synchronize) thông tin của đối tượng dữ liệu này trên toàn bộ các node của hệ thống. Lớp `MNode` (đã được cài đặt sẵn) biểu diễn 1 node xử lý tính toán trong hệ thống.

Hãy sửa lại cách cài đặt phương thức cập nhật giá trị thuộc tính có tên là `strAttributeName` trong đối tượng thuộc lớp `MObject` (ở Câu 5) để mỗi khi cập nhật giá trị thuộc tính, đối tượng `MObject` luôn tự động gọi đến phương thức `UpdateObject()` của lớp `DecentralizedAppEngine`. Tham số input của hàm `UpdateObject()` có dạng chuỗi chính là `ID` của đối tượng `MObject` cần cập nhật.

Câu 9. Đồng bộ hóa đối tượng dữ liệu trên mọi node xử lý và lưu trữ **(0,5 điểm)**

Hãy cài đặt phương thức `UpdateObject()` của lớp `DecentralizedAppEngine` để cập nhật thông tin của đối tượng `MObject` (với `ID` cho trước) trên tất cả các node, giả sử đã có sẵn hàm `SynchronizeObject()` của lớp `MNode` (với tham số input có dạng chuỗi chính là `ID` của đối tượng `MObject` cần cập nhật).

Câu 10. Xử lý phi tập trung 1 giao tác trên tất cả mọi node xử lý và lưu trữ **(0,5 điểm)**

Khi người dùng có tài khoản với ID là `AccountID` muốn thực thi 1 giao tác có ID là `TransactionID`, người dùng sẽ gọi phương thức tĩnh `ProcesMTransaction()` của lớp `DecentralizedAppEngine`.

Giải sử phương thức này đã được cài đặt như sau:

```
public string ProcesMTransaction(string AccountID, string TransactionID,
                                  string[] input) {
    List<MObject> candidates = new List<MObject>();
    foreach (string nodeID in nodes.Keys) {
        MObject localResult = nodes[nodeID].ProcesMTransaction(
                              AccountID, TransactionID, input);
        candidates.Add(localResult);
    }
    MObject result = FindMostReliableResult(candidates);
    if (result == null) return "";
    return result.ID;
}
```

a) Hãy giải thích ý nghĩa của việc cài đặt hàm trên. Lưu ý mảng `candidates` và hàm `FindMostReliableResult()`.

b) Hãy đề xuất 1 giải pháp đơn giản để cài đặt hàm `FindMostReliableResult()`.

PHẦN II – TRẢI NGHIỆM THỰC TẾ

Câu 11 – TÌM HIỂU VÀ TRẢI NGHIỆM THỰC TẾ **(1,0+ điểm)**

Hãy trình bày một ví dụ cụ thể về việc ứng dụng kiến trúc phần mềm trong đồ án hay đề tài mà anh/chị ***đã thực hiện*** hoặc ***đã tìm hiểu***. Cần trình bày rõ: (a) tình huống, yêu cầu thực tế; (b) giải pháp đề xuất; (c) ưu điểm và hạn chế của giải pháp.

CHÚC CÁC ANH CHỊ LÀM BÀI THI TỐT!

-HẾT-