CUI

Comsats University Islamabad (Vehari Campus)

Academic Year 22-27
Department: CS

Topic of Assignment:

"Refactoring"

Student Name:

Asadullah
(FA22-BSE-037)

Subject:

Software Re-Engineering

Batch: “19”

Teacher Name:

Dr.Manzoor Ahmad

Program #1:

Before Refactoring:

```
class Account {
 float principal, rate;
 int daysActive, accountType;
 public static final int STANDARD = 0;
 public static final int BUDGET = 1;
 public static final int PREMIUM = 2;
 public static final int PREMIUM_PLUS = 3;
}

float calculateFee(Account accounts[]) {
 float totalFee = 0;
 Account account;
 for (int i = 0; i < accounts.length; i++) {
   account = accounts[i];
   if (account.accountType == Account.PREMIUM ||
       account.accountType == Account.PREMIUM_PLUS) {
     totalFee += .0125 * (account.principal *
         Math.exp(account.rate * (account.daysActive / 365.25))
         - account.principal);
   }
 }
 return totalFee;
}
```

After Refactoring:

```
class Account {
    // Refactored: encapsulated fields
    private float principal, rate;
    private int daysActive;
    private int accountType;

    // Constants used instead of magic numbers
    public static final int STANDARD = 0;
    public static final int BUDGET = 1;
    public static final int PREMIUM = 2;
    public static final int PREMIUM_PLUS = 3;
    // Refactored: moved condition into method
    public boolean isPremium() {
        return accountType == PREMIUM || accountType == PREMIUM_PLUS;
    }
    // Refactored: moved calculation into method
    public float calculateInterest() {
        return 0.0125f * (principal * (float)Math.exp(rate * (daysActive / 365.25)) - principal);
    }
}
float calculateFee(Account[] accounts) {
    float totalFee = 0;
    for (Account account : accounts) {
        if (account.isPremium()) {
            totalFee += account.calculateInterest(); // Delegated logic
        }
    }
    return totalFee;
}
```

Program #2:

Before Refactoring:

```
class Buffer {
public:
 Buffer(size_t size) : size(size), data(new char[size]) {}
 ~Buffer() { delete[] data; }

 // Copy constructor
 Buffer(const Buffer& other) : size(other.size), data(new char[other.size]) {
   std::copy(other.data, other.data + other.size, data);
 }
 // Copy assignment
 Buffer& operator=(const Buffer& other) {
   if (this != &other) {
     delete[] data;
     size = other.size;
     data = new char[other.size];
     std::copy(other.data, other.data + other.size, data);
   }
   return *this;
 }
private:
 size_t size;
 char* data;
};
```

After Refactoring:

```
class Buffer {
public:
    Buffer(size_t size) : size(size), data(new char[size]) {}
    ~Buffer() { delete[] data; }

    // Copy constructor
    Buffer(const Buffer& other) : size(other.size), data(new char[other.size]) {
        std::copy(other.data, other.data + other.size, data);
    }
    // Copy assignment operator
    Buffer& operator=(const Buffer& other) {
        if (this != &other) {
            delete[] data;
            size = other.size;
            data = new char[other.size];
            std::copy(other.data, other.data + other.size, data);
        }
        return *this;
    }
    //  Move constructor
    Buffer(Buffer&& other) noexcept : size(other.size), data(other.data) {
        other.size = 0;
        other.data = nullptr;
    }
    //  Move assignment operator
    Buffer& operator=(Buffer&& other) noexcept {
        if (this != &other) {
            delete[] data;
            size = other.size;
            data = other.data;
            other.size = 0;
            other.data = nullptr;
        }
        return *this;
```

```
    }
private:
    size_t size;
    char* data;
};
```

Program #3:

Before Refactoring:

```
public findOriginalHorizontal(): number {
 let retval = 0;
 let reflectline = 0;
 let mapSeen: string[] = [];
 mapSeen.push(this.map[0]);
 for (let i = 1; i < this.map.length; i++) {
   mapSeen.push(this.map[i]);
   if (reflectline == 0 && (this.map[i] == mapSeen[i - 1])) {
     reflectline = i;
     continue;
   }
   if (reflectline > 0) {
     let diff = i - reflectline;
     if (reflectline - diff - 1 >= 0) {
       if (mapSeen[reflectline - diff - 1] != this.map[i]) {
         reflectline = 0;
       }
     }
   }
 }
 return reflectline;
}

public findOriginalVertical(): number {
 let retval = 0;
 let reflectline = 0;
 let mapSeen: string[] = [];
 const rotatedMap = rotate(this.map);
 mapSeen.push(rotatedMap[0]);
 for (let i = 1; i < rotatedMap.length; i++) {
   mapSeen.push(rotatedMap[i]);
   if (reflectline == 0 && (rotatedMap[i] == mapSeen[i - 1])) {
     reflectline = i;
     continue;
   }
   if (reflectline > 0) {
     let diff = i - reflectline;
     if (reflectline - diff - 1 >= 0) {
       if (mapSeen[reflectline - diff - 1] != rotatedMap[i]) {
         reflectline = 0;
       }
     }
   }
 }
 return reflectline;
}
```

After Refactoring (DRY Applied):

```
//  Common logic extracted to avoid repetition
private findOriginal(reflectData: string[]): number {
    let reflectline = 0;
    let mapSeen: string[] = [reflectData[0]];

    for (let i = 1; i < reflectData.length; i++) {
        mapSeen.push(reflectData[i]);
        if (reflectline === 0 && reflectData[i] === mapSeen[i - 1]) {
            reflectline = i;
            continue;
        }
        if (reflectline > 0) {
            let diff = i - reflectline;
            if (reflectline - diff - 1 >= 0) {
                if (mapSeen[reflectline - diff - 1] !== reflectData[i]) {
                    reflectline = 0;
                }
            }
        }
    }
    return reflectline;
}
//  Clean and DRY interface
public findOriginalHorizontal(): number {
    return this.findOriginal(this.map);
}
public findOriginalVertical(): number {
    return this.findOriginal(rotate(this.map));
}
```