SE302 – ПХ БҮТЭЭЛТ
Лекц 5

Class

- Анх програмыг илэрхийллүүд зүйл анги
- 1970, 1980 онд функцуудын зүйл анги
- XXI зуунд классуудын зүйл анги
  - мэдээлэл болон функцуудын цуглуулга

- Сайн чанартай класс хэрхэн үүсгэх

Class Foundations: Abstract Data Types

- Хийсвэр өгөгдлийн төрөл нь өгөгдөл болон тэрхүү өгөгдөл дээр ажиллах функцуудын цуглуулга
- ADT ойлгохгүйгээр програмистууд класс нэртэй зүйл л үүсгэнэ
  - Хоорондоо холбоотой өгөгдөл болон функцуудын цуглуулга
  - Классыг ойлгосноор
    - анхнаасаа хялбар хэрэгжүүлж
    - цаашид шинэчлэхэд хялбар болно

Example of the Need for an ADT

- Текст гаралт гаргадаг програмын хувьд
  - фонт, хэмжээ, аттрибут (bold, italic)
  - ADT ашиглавал эдгээр өгөгдөлтэй ажилладаг багц функцууд байна
  - Фонттой ажилладаг функц болон өгөгдлийн цугууллага нь ADT юм
- Хэрэв ADT ашиглахгүй бол
  - Ad hoc хандлагаар фонтуудыг удирдана
  - Фонтын хэмжээг 12 pt болгох (16px)
    - currentFont.size = 16

Example of the Need for an ADT

- Хэрэв сангийн функц ашиглавал
  - currentFont.size = PointsToPixels(12)
- Аттрибутын нэрийг онцгой байдлаар өгвөл
  - currentFont.sizeInPixels = PointsToPixels(12)
- Дараах хоёр аттрибут байх боломжгүй
  - currentFont.sizeInPixels
  - currentFont.sizeInPoints
- Хэрэв фонтын хэмжээг зарим газар өөрчлөх хэрэг гарвал
  - Програмын бусад хэсэгт үүнтэй төстэй кодын хэсэг олдоно

Example of the Need for an ADT

Хэрэв фонтыг Bold болгох болвол

0x02 тогтмолтой логик OR үйлдэл хийнэ

currentFont.attribute = currentFont.attribute or 0x02

Хэрэв та сайн ажиллавал

currentFont.attribute = currentFont.attribute or BOLD

currentFont.bold = True

Дээрх байдлаар програм бичвэл эдгээртэй ижил мөр програм дээр маш их үүснэ

Benefits of Using ADTs

- ad-hoc хандлага бол програмчлалын муу дадал
- Сайн програмчлалын дадалууд нь
  - Гүйцэтгэлийн дэлгэрэнгүй хэсгийг нуух
    - Өгөгдлийн төрөл өөрчлөгдөхөд
      - ADT-д гүйцэтгэлийн дэлгэрэнгүйг нуух
  - Өөрчлөлт нь програмд бүхэлд нь нөлөөлөхгүй байх
    - фонтыг илүү баялаг болгох
      - superscript, дээгүүр нь дарах
      - Өөрчлөлтийг нэг газар хийнэ
      - өөрчлөлт програмд нөлөөлөхгүй
  - Интерфэйсийг илүү утга учиртай болгох
    - Хоёрдмол утгагүй байх - point, pixel
    - currentFont.size = 16
    - point, pixel -ээр өөрчлөх функцууд аль аль нь байх

Benefits of Using ADTs

- Програмыг илүү тод, зөв болгох
  - replace хийж засах
    - currentFont.attribute = currentFont.attribute or 0x02
    - currentFont.SetBoldOn()
  - буруу бүтэцийн нэр, буруу талбарын нэр, буруу үйлдэл, буруу утга
    - 0x02 -ын оронд 0x20 байх
- Програм нь өөрийгөө илүү баримтжуулсан байх
  - rentFont.attribute or 0x02
    - 0x02 -> BOLD
  - currentFont.SetBoldOn()

Benefits of Using ADTs

- Програм даяар параметр дамжуулахгүй байх
  - Одоогын фонтыг өөрчлөх эсвэл фонттой ажиллах функц бүрт дамжуулах шаардлагагүй
  - currentFont
    - параметр болгож дамжуулахгүй
    - глобал хувьсагч болгохгүй
  - Функц нь ADT-ын нэг хэсэг учраас өгөгдөл байхгүй гэж санаа зовохгүй

Benefits of Using ADTs

- Та доод түвшний гүйцэтгэлээс илүү бодит амьдрал дээрх объекттэй ажиллах чадвартай
  - Програмд фонттой ажиллах үйлдлүүдийг тодорхойлох
  - Гүйцэтгэл нь сонин биш
    - массивт хандах, бүтэц тодорхойлох, үнэн эсвэл худал

```
currentFont.SetSizeInPoints( sizeInPoints )
currentFont.SetSizeInPixels( sizeInPixels )
currentFont.SetBoldOn()
currentFont.SetBoldoff()
currentFont.SetItalicOn()
currentFont.SetItalicoff()
currentFont.SetTypeFace( faceName )
```

More Examples of ADTs

Set of Help Screens

Add help topic
Remove help topic
Set current help topic
Display help screen
Remove help display
Display help index
Back up to previous screen

Pointer

Get pointer to new memory
Dispose of memory from existing pointer
Change amount of memory allocated

Menu

Start new menu
Delete menu
Add menu item
Remove menu item
Activate menu item
Deactivate menu item
Display menu
Hide menu
Get menu choice

File

Open file
Read file
Write file
Set current file location
Close file

Elevator

Move up one floor
Move down one floor
Move to specific floor
Report current floor
Return to home floor

Good Class Interfaces

- Сайн чанартай класс үүсгэхэд сайн интерфэйс үүсгэх хэрэгтэй
- Энэ нь сайн хийсвэрлэлийг үүсгэх хэрэгтэй бөгөөд дэлгэрэнгүй хэсэг хийсвэрлэлийн ард нуугдаж орхигдоно.

Good Abstraction

C++ Example of a Class Interface That Presents a Good Abstraction

```
class Employee {
public:
   // public constructors and destructors
   Employee();
   Employee(
      FullName name,
      String address,
      String workPhone,
      String homePhone,
      TaxId taxIdNumber,
      JobClassification jobClass
   );

   // public routines
   FullName GetName() const;
   String GetAddress() const;
   String GetWorkPhone() const;
   String GetHomePhone() const;
   TaxId GetTaxIdNumber() const;
   JobClassification GetJobClassification() const;
   ...
private:
   ...
};
```

Good Abstraction

- Классын бүх нэмэлт өгөгдөл болон функцуудыг мэдэх шаардлагагүй
- Классын интерфэйсийн хийсвэрлэл нь тогтвортой байх шаардлагатай

Good Abstraction

C++ Example of a Class Interface That Presents a Poor Abstraction

```
class Program {
public:
   ...
   // public routines
   void InitializeCommandStack();
   void PushCommand( Command command );
   Command PopCommand();
   void ShutdownCommandStack();
   void InitializeReportFormatting();
   void FormatReport( Report report );
   void PrintReport( Report report );
   void InitializeGlobalData();
   void ShutdownGlobalData();
   ...
private:
   ...
};
```

Good Abstraction

- Хамаарлыг нь ойлгоход төвөгтэй
  - Command stack
  - Print report
  - Global data
- Функцууд нь классын зорилго руу төвлөрсөн байхаар зохион байгуулах

Good Abstraction

- Классын public функцууд дээр тулгуурлаж классын хийсвэрлэлийн өөрчлөлтийг хийсэн

C++ Example of a Class Interface That Presents a Better Abstraction

```
class Program {
public:
   ...
   // public routines
   void InitializeUserInterface();
   void ShutDownUserInterface();
   void InitializeReports();
   void ShutDownReports();
   ...
private:
   ...
};
```

Good Abstraction

- Хамааралгүй өгөгдлүүдийг өөр класс руу зөөх
  - зарим өгөгдөл дээр зарим функц ажилладаг
  - үлдсэн зарим дээр нь үлдсэн функцууд ажилладаг
- Яг үнэндээ 2 класс нэг классын баг зүүсэн байна гэсэн үг

Good Abstraction

- Хэрэв боломжтой бол Programmatic интерфэйс хийх нь Semantic интерфэйсээс илүү дээр
  - Programmatic интерфэйс
    - өгөгдлийн төрөл, аттрибут
    - компайлер мөрдүүлэх боломжтой
  - Semantic интерфэйс
    - Интерфэйс хэрхэн хэрэглэгдэх тухай
    - компайлер мөрдүүлэх боломжгүй
    - RoutineA нь RoutineB-ээс өмнө дуудагдах ёстой
    - RoutineA руу datamember1-ийг ачааллаж дамжуулахгүй бол тухайн функц алдаа өгнө
    - Тайлбараар баримтжих ёстой
    - Programmatic интерфэйс болгох арга замыг хайх хэрэгтэй

Good Encapsulation

- Класс болон гишүүд рүү хандах хандалтыг бууруулах
  - Encapsulation дэмжиж дизайн гаргахад чиглэсэн хэд хэдэн дүрмүүдийн нэг
  - Илүү ихийг далдлах нь багыг далдалсанаас ерөнхийдөө сайн
- Классын интерфэйс дээр дэлгэрэнгүй гүйцэтгэлийг хувийн хандлатаар зааж өгөх
  - програмистууд харах шаардлага байхгүй

Containment (“has a” Relationships)

- “has a” харилцаа
  - Employee
    - name, phone number, taxID
  - Гишүүн өгөгдөл нь долоогоос дээш бол аюултай
    - олон жижиг классуудад хуваах
  - Гишүүн өгөгдөл нь өгөгдлийн үндсэн төрөл
    - 7 + 2
      - Алдаа гарах эрсдэл нь ихэснэ
  - Гишүүн өгөгдөл нь объект үед
    - 7 - 2

Inheritance (“is a” Relationships)

- Нэг класс нь өөр классыг тодорхойлох санаа
- Ерөнхий элемэнтүүдтэй суурь классыг тодорхойлох зорилготой
- Удамшил ашиглахаар шийдсэн бол
  - Гишүүн функц бүр өвлөн авах классад харагдах уу?
    - Өгөгдмөл гүйцэтгэл гэж бий юу?
  - Гишүүн өгөгдөл бүр өвлөн авах классад харагдах уу?

Inheritance (“is a” Relationships)

- Удамшил нь програмын төвөгтэй цогц байдлыг нэмэгдүүлдэг
  - үүнийг хязгаарлах хэрэгтэй
  - c++ - non-virtual, java - final, non-overrideable - VB
- Ерөнхийн интерфэйсийн гишүүн өгөгдөл болон гишүүн функцуудыг удамшлын харилцаанд аль боломжтой байхаар дээшээ зөөх
  - Ерөнхийлөл
  - Нарийвчлал

Inheritance (“is a” Relationships)

- Даран тодорхойлох боломжгүй функцыг даран тодорхойлохгүй байх
  - эцэг классын private функцыг хүү классад тодорхойлох, ижил нэртэйгээр
- Хэтэрхий гүн удамшлын харилцаанаас зайлсхийх
  - алдааны үзүүлэлтыг нэмэгдүүлдэг
  - хамгийн ихдээ 6 байх
- Зөвхөн нэг класс удамшдаг үндсэн класс
  - сэжигтэй класс
  - Магадгүй нэг өдөр хэрэгтэй болно (ирээдүйд)
  - Үнэхээр шаардлагатай бол үүсгэх

Inheritance (“is a” Relationships)

- Төрөл шалгахаас полиморфизмыг илүүд үзэх
  - Заримдаа олон давтагдсан case-ийн оронд Polymorphism илүү сайн шийдлийг бий болгодог

C++ Example of a Case Statement That Probably Should Be Replaced by Polymorphism

```
switch ( shape.type ) {
   case Shape_Circle:
      shape.DrawCircle();
      break;
   case Shape_Square:
      shape.DrawSquare();
      break;
   ...
}
```

Inheritance (“is a” Relationships)

C++ Example of a Case Statement That Probably Should Not Be Replaced by Polymorphism

```
switch ( ui.Command() ) {
   case Command_OpenFile:
      OpenFile();
      break;
   case Command_Print:
      Print();
      break;
   case Command_Save:
      Save();
      break;
   case Command_Exit:
      ShutDown();
      break;
   ...
}
```

Inheritance (“is a” Relationships)

- Бүх өгөгдлийг private болго, protected биш
  - Joshua Bloch
    - Удамшил нь Encapsulation-ыг зогсоодог
  - Үнэхээр эцэг классын гишүүн өгөгдөлд хандах шаардлага гарвал
    - protected хандагч функцуудээр хангаж өг

Member Functions and Data

- Класс дах гишүүн функцуудыг аль болох бага байлгах
  - Гишүүн функцын тоо их байх нь алдаа гарах үзүүлэлттэй хамааралтай
- Шууд бус функц дуудалтыг бууруулах
  - account.ContactPerson() - ok
  - account.ContactPerson().DaytimeContactInfo()

Reasons to Create a Class

- Бодит амьдрал дээрх объект
  - Сайн шалтгаан байсаар байгаа
- Хийсвэр объект
  - Бодит амьдрал дээр объект нь байддаггүй
  - Дүрс
    - Тойрог, тэгш өнцөгт
- Төвөгтэй байдлыг бууруулах
  - Хамгийн чухал шалтгаануудын нэг
  - Класс үүсгэж мэдээллийг нууцалсанаар түүний талаар бодох шаардлагагүй болно
    - Тухайн классыг бичихтэй л энэ талаар бодно
    - Бичсэний дараа мартаж болох бөгөөд дотоод ажил нь яаж хийгдсэнийг дахиж бодохгүй байж болно

Reasons to Create a Class

- Төвөгтэй байдлыг салгана
  - Цөгц алгоритм
  - их хэмжээний өгөгдөлтэй харьцах
  - ээдрээтэй харилцааны протокол
    - Хэрэв алдаа гарвал хайхад хялбар
    - Хийсэн засвар бусад кодонд огтхон ч хүрэхгүй
    - хуучин алгоритмыг сайжуулахад амар
- Гүйцэтгэлийн дэлгэрэнгүйг нуух
  - Класс үүсгэх гайхалтай шалтгаан
- Өөрчлөлтийн нөлөөг хязгаарлах
  - Өөр бусад классад хийгдсэн өөрчлөлт нөлөөлөхгүй

Reasons to Create a Class

- Параметр дамжуулалт
  - Параметрыг хэд хэдэн функцээр дамжуулж байвал
    - Тэдгээр функцууд нь нэг классад
    - Параметр нь тухайн классын гишүүн өгөгдөл байх
- Удирдлагын нэгдсэн төв хийх
  - Төхөөрөмжүүдийн удирдлага
    - file, database connection, printer
  - Мэдээлэл нууцлалттай төстэй
- Дахин ашиглалтыг дэмжих
  - Сайн бичигдсэн класс
  - Өөр програмд ашиглаж болох, embedded

Classes to Avoid

- Бурхан класс үүсгэхээс зайлсхийх
  - Бүгдийг мэддэг
  - Бүгдийг хийдэг
  - Ихэнх цагаа Get, Set функц дээр зарцуулах
- Зохицоогүй класс
  - Зөвхөн гишүүн өгөгдлөөс тогтох, функцгүй
  - өөрөөсөө асуух
    - Нэг эсвэл өөр олон классын гишүүн өгөгдлүүд болох
- Үйл үгээр нэрлэгдсэн класс
  - Энэ бол зөвхөн функц
    - яг үнэндээ класс биш
    - DatabaseInitialization()
    - String-Builder()