Topic 13 – Object Oriented Programming (OOP)

Part 2 – Designing

Where to begin…

Before we begin to program we need to analyze the project and design the program

1. Identify the Object Types
2. Identify the attributes
3. Identify the methods
4. Design the methods
5. Design the program

Identifying Object Types

- First, we need to think about the object types we will need
- One approach is to identifying possible object types is to extract the **nouns** from the project description

For example

*A program is required that will assist with the **administration** of the **farm**. The **farm** is situated in **Fallbrook**. **Farmer Pete** owns the **farm** and together with his **family** takes care of all the **animals**. There are many **sheep**, many pigs and two **horses** on the **farm**. The **farmer** also has a **dog**, called **Buddy**. The **farmer** wants to be able to keep track of all of his **animals**; he needs to know the **name**, **age** and **approximate value** of each. When he sells an **animal** he needs to be able to record this **fact** together with the **name** of the **farmer** he has sold the **animal** to. He never sells his **horses** or his **dog**.*

Identifying Object Types

- First, we need to think about the object types we will need
- One approach is to identifying possible object types is to extract the **nouns** from the project description

For example

*A program is required that will assist with the **administration** of the **farm**. The **farm** is situated in **Fallbrook**. **Farmer Pete** owns the **farm** and together with his **family** takes care of all the **animals**. There are many **sheep**, many pigs and two **horses** on the **farm**. The **farmer** also has a **dog**, called **Buddy**. The **farmer** wants to be able to keep track of all of his **animals**; he needs to know the **name**, **age** and **approximate value** of each. When he sells an **animal** he needs to be able to record this **fact** together with the **name** of the **farmer** he has sold the **animal** to. He never sells his **horses** or his **dog**.*

animals	*age*
sheep	*approximate value*
pigs	*animal*

Identifying Object Types

- Now, we want to shorten the list
  - First clarify the names – remove unnecessary plurals
  - Remove duplicates

For example

program	*horses*	*fact*
Farmer Pete's farm	*Farmer Pete's farm*	*farmer buying an animal*
administration	*Farmer Pete*	*farmer buying an animal*
Farmer Pete's farm	*dog*	*animal*
Fallbrook	*Buddy*	*horses*
Farmer Pete	*Farmer Pete*	*dog*
family	*animals*	
Farmer Pete's farm	*name*	
animals	*age*	
sheep	*approximate value*	
pigs	*animal*	

Identifying Object Types

- Now, we want to shorten the list
  - First clarify the names – remove unnecessary plurals
  - Remove duplicates
  - Eliminate object types that can be obviously eliminated object types
  - Remove Items mentioned in passing like Farmer Pete and his farm, family, & etc.

For example

program	*horse*	*fact*
Farmer Pete's farm	*age*	*farmer buying an animal*
administration	*approximate value*	
Fallbrook	*dog*	
Farmer Pete	*farmer buying an animal*	
Farmer Pete		
family		
sheep	*name*	
pigs	*age*	
sheep	*approximate value*	
dog		

Identifying Object Types

- Now, we want to shorten the list
  - First clarify the names – remove unnecessary plurals
  - Remove duplicates
  - Eliminate object types that can be obviously eliminated object types
  - Remove Items mentioned in passing like Farmer Pete and his farm, family, & etc.

For example

~~*animal*~~
sheep
pig
~~*farmer*~~ *buying an animal*
~~*dog*~~
~~*name*~~
~~*animal*~~
~~*approximate value*~~
farmer buying an animal
horse
dog

name
age → sheep
approximate value → pig
farmer buying an animal → buyer

- Some of these are good candidates for attributes, but not necessarily object types – like name, age, & etc.
- Since the farmer will never sell his dog & horses we can remove those too
- animal is just a generic name for our animals – we could use this unless there were different attributes
- Our final step is to name our object types
  - Singular

Identifying The Attributes

- Now that we have our object types – we need to think of the attributes
  - Things that will differentiate each instance of our object type

OBJECT TYPE	ATTRIBUTE	TYPE
Sheep	name	string
	age	integer
	value	float
	buyer	Farmer
Pig	name	string
	age	integer
	value	float
	buyer	Farmer
Farmer	name	string
	address	string
	phone number	string

Identifying Methods

Now we need to think of which transactions or events that are described in the problem that will change the values of our attributes

- In our problem the farmer *sells an animal*
- Also, what attributes does the farmer need to know?
  - How do we provide this information?
- Name, age, and approximate value
  - We have to have a way to display these for each animal
- Think also of which attributes will likely change
  - Age & approximate value for the animals
- For all attributes we need a way to set their initial values
- The farmer may want a list of all animals
  - This would be out of the context of a single object and would be handled by the program using the object

Identifying the Methods

OBJECT TYPE	METHOD
Sheep	Constructor
	Destructor
	SetInitialValues
	ChangeAge
	ChangeValue
	Display
	GetName
	GetAge
	GetValue
	Sell
Pig	Constructor
	Destructor
	SetInitialValues
	ChangeAge
	ChangeValue
	Display
	GetName
	GetAge
	GetValue
	Sell
Farmer	Constructor
	Destructor
	SetInitialValues

Designing the Methods

We want to detail the following about each method

- Purpose
- Inputs
- Values changed
- effects → output to the screen or data file

Sheep OBJECT TYPE

METHOD	INPUT	INPUT TYPE	RETURN TYPE
Constructor	*<none>*		
Destructor	*<none>*		
SetInitialVaues	name	string	void
	age	int	
	value	float	
ChangeAge	age	int	void
ChangeValue	value	float	void
Sell	buyer	Farmer	void
Display	*<none>*		void
GetName	*<none>*		string
GetAge	*<none>*		int
GetValue	*<none>*		float

NOTE: We would also include descriptions of each of these and specify output

Conversations with the farmer

- At some point we would want to clarify some things with the farmer (hopefully before this point)
- First, does he anticipate any differences in pigs vs. sheep
  - If not, maybe we should have an animal object with an attribute that differentiates the animal type
- Let's say the farmer says no, he sells all livestock as pets so other attributes won't apply
- Given this information we decide to have one Object Type
  - Animal
  - And add an attribute to differentiate pigs from sheep
- Another alternative is to use *inheritance* to create pig and sheep classes deriving from the animal class
  - We will be learning this next

Putting All Together Our Animal Class

```
class Animal
{
   public:
            Animal ();
            ~Animal ();

            void SetInitialValues (string aName, string aType, int aAge, float aValue);
            void ChangeAge (int aAge);
            void ChangeValue (float aValue);
            void Sell (Farmer aBuyer);

            void Display () const;
            string GetName () const;
            string GetType () const;
            int GetAge () const;
            float GetValue () const;

   private:
            string   name;
            string   type;
            int      age;
            float    value;
            Farmer buyer;
};
```

Exercise

- You have been hired as a programmer to define a class that implements the 24-hour time of day (a clock). This class will be used as part of a large program, so you need to clearly define the class interface
  - You will need to:
    - Identify the Object Type(s)
    - Identify the Attributes
    - Identify the Methods
  - Once the class interface is designed
    - Design and implement the methods
    - Design and implement the program

ClockType Class

- Attributes
  - To represent time of day we need three integer numbers
    - hour (0-23) – for a 24-hour clock
    - minute (0-59)
    - second (0-59)
  - These numbers should be only accessed within the class
    - private data
  - They should be initialized as zero
    - default value zero (=0)

ClockType Class

Methods

There are many possible methods to manipulate time of day. After reviewing with the programming team the following methods are needed:

- Set time
  - input: hour, minute, second
  - return: none
  - type: mutator
- Retrieve time
  - input: hour, minute, second (as reference to return the values)
  - return: none
  - type: accessor

ClockType Class

- Print time (format “hh:mm:ss”)
  - input: none
  - return: none
  - type: accessor
- Increment time by one unit
  - input: (type =1 for hour, =2 for minute, =3 for second)
  - return: none
  - type: mutator
- Decrement time by one unit
  - input: (type =1 for hour, =2 for minute, =3 for second)
  - return: none
  - type: mutator

ClockType Class

- Compare two times for equality
  - input: ClockType (as a constant reference)
  - return: yes(=1) or no(=0)
  - type: accessor
- Constructor (initialize attributes =0)
  - input: none
  - Output: none
  - type: constructor

- All methods should be accessible from outside of the class
  - public data

```
class ClockType
{
    public:
        ClockType ();
        ~ClockType ();
        void setTime (int hour, int minute, int second);
        void getTime (int& hour, int& minute, int& second) const;
        void printTime () const;
        void incrementTime (int type);
        void decrementTime (int type);
        bool equalTime (const ClockType& otherClock) const;

    private:
        int hr;
        int min;
        int sec;
};
```