Welcome

Object Oriented Programming with C++
CIS 265

Week 10 – Putting it together

Christopher K. Burns

Schedule

Week		Content
1	1/11	*Review* Chapter 1 Intro to Computers and C++ Programming Chapter 2 Control Structures Chapter 3 Functions Chapter 4 Arrays
2	1/18	Additional Array and Function Topics Chapter 5 Pointers and Strings Lab 1 – Functions, Arrays, and Strings Homework 1 Assigned
3	1/25	Chapter 6 Classes and Data Abstraction
4	2/1	Chapter 7 Classes: Part I Lab 2 – Classes 1 Homework 1 Due
5	2/8	***MID-TERM EXAMINATION*** *(Chapters 1 through 7*)* **only portions of chapter 7 that were covered in class* *Final Project Assigned*
6	2/15	Chapter 7 Classes: Part II Chapter 8 Operator Overloading Homework 2 Assigned
7	2/22	Chapter 8 Operator Overloading Homework 2 Due Lab 3 – Classes 2
8	3/1	Chapter 9 Inheritance: Part I Homework 3 Assigned
9	3/8	Chapter 9 Inheritance: Part II Chapter 10 Polymorphism Homework 3 Due
10	3/15	Chapter 10 Polymorphism Lab 4 – Inheritance and Polymorphism
11	3/22	***FINAL PROJECTS DUE***

Agenda

Tonight's agenda

- Review
- Getting user input
- More on file streams
- Strategies for class design
- Any questions?
- Final Project Lab

Object Oriented Programming

Home Work

Final Project!!!

Final Project

Final Project must be turned in before March 27

You may turn it in as early as tonight.

Classes - Review

What we've covered about classes up to this point:

- Basic structure of a class
- Constructors and overloading
- Separating Interface from Implementation
- Data encapsulation (get/set)
- Static members
- this pointer
- Operator Overloading
- Standard C++ Library – String and Vector
- Inheritance
- Polymorphism
- Templates

Polymorphism - Review

Key points on polymorphism:

- Objects can be referred to as their base classes (e.g. Dog and Cats can be referred to as Animals)
- Derived class can override their base classes methods. When overriding methods and using polymorphism, remember that the base classes method must be virtual.
- If you have an array of polymorphic objects, you can find out what they really are by using “Run Time Type Info”.

Polymorphism – base methods virtual

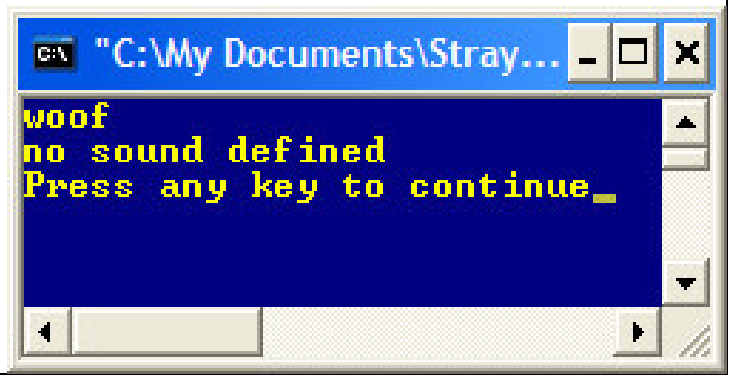
```
class Animal
{
public:
  string name;
  virtual void produceSound() { cout << "no sound defined\n"; }
};

class Dog : public Animal
{
public:
  Dog() { name = "Dog"; }
  void produceSound() { cout << "woof\n"; }
};

class Cat : public Animal
{
public:
  Cat() { name = "Cat"; }
};

void main()
{
  Animal* animals[2];
  animals[0] = new Dog();
  animals[1] = new Cat();

  for( int i = 0; i < 2; i++ )
  {
    animals[i]->produceSound();
  }
}
```



Polymorphism – using type id

```
void main()
{
  Animal* animals[2];
  animals[0] = new Dog();
  animals[1] = new Cat();

  for( int i = 0; i < 2; i++ )
  {
    cout << typeid( *animals[i] ).name() << " says ";
    animals[i]->produceSound();
    if ( typeid( *animals[i] ) == typeid( Cat ) )
    {
      dynamic_cast< Cat* >(animals[i])->scratch();
    }
  }
}
```

What do I do now?

Inheritance and polymorphism are nice, but how can I make a usable program?

- Strategies on user input
- Saving and loading classes from a file
- Designing classes - methodologies

Writing a program

Most modern programs have GUIs (Graphical User Interface) or HMIs (Human Machine Interface) that are based on Windows (e.g. buttons, dialogs, edit boxes…)

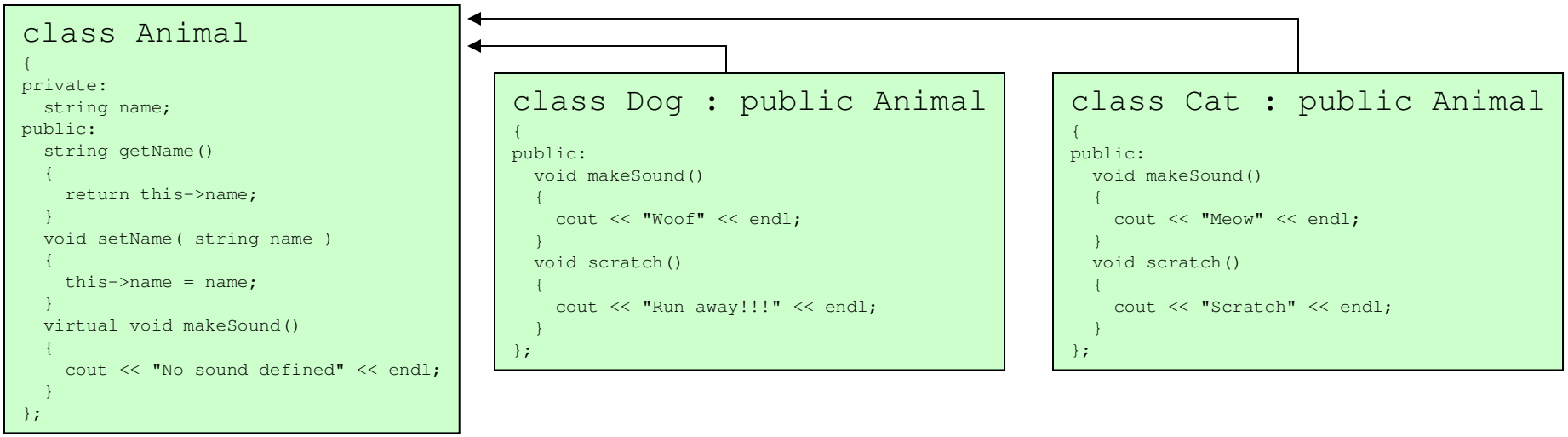
Before there were Windowing environments, there was the console… these typically use menus or command prompts to interact with users.

In CIS 265, we'll stick with the old-school style text based user interface.

The Basic User Menu

Let's create a simple program that let's the user enter some data.

Here are the classes we're going to use:



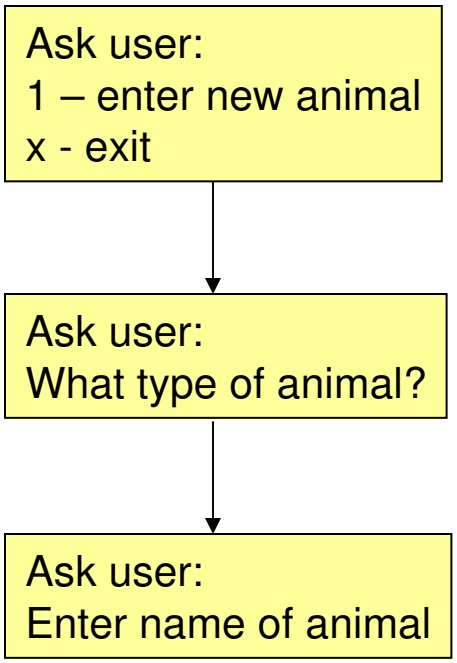
```
class Animal
{
private:
  string name;
public:
  string getName()
  {
    return this->name;
  }
  void setName( string name )
  {
    this->name = name;
  }
  virtual void makeSound()
  {
    cout << "No sound defined" << endl;
  }
};
```

```
class Dog : public Animal
{
public:
  void makeSound()
  {
    cout << "Woof" << endl;
  }
  void scratch()
  {
    cout << "Run away!!!" << endl;
  }
};
```

```
class Cat : public Animal
{
public:
  void makeSound()
  {
    cout << "Meow" << endl;
  }
  void scratch()
  {
    cout << "Scratch" << endl;
  }
};
```

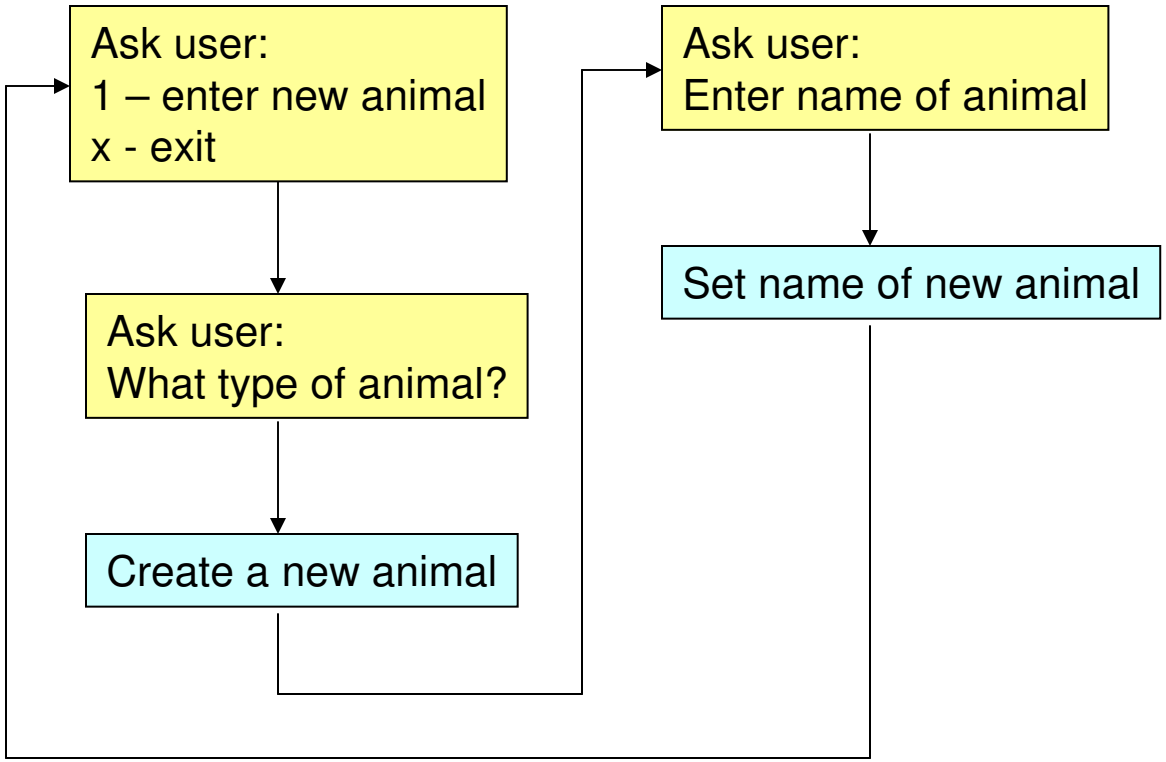
The Basic User Menu

What do we want our program to do?



The Basic User Menu

What do we want our program to do?



Our menu

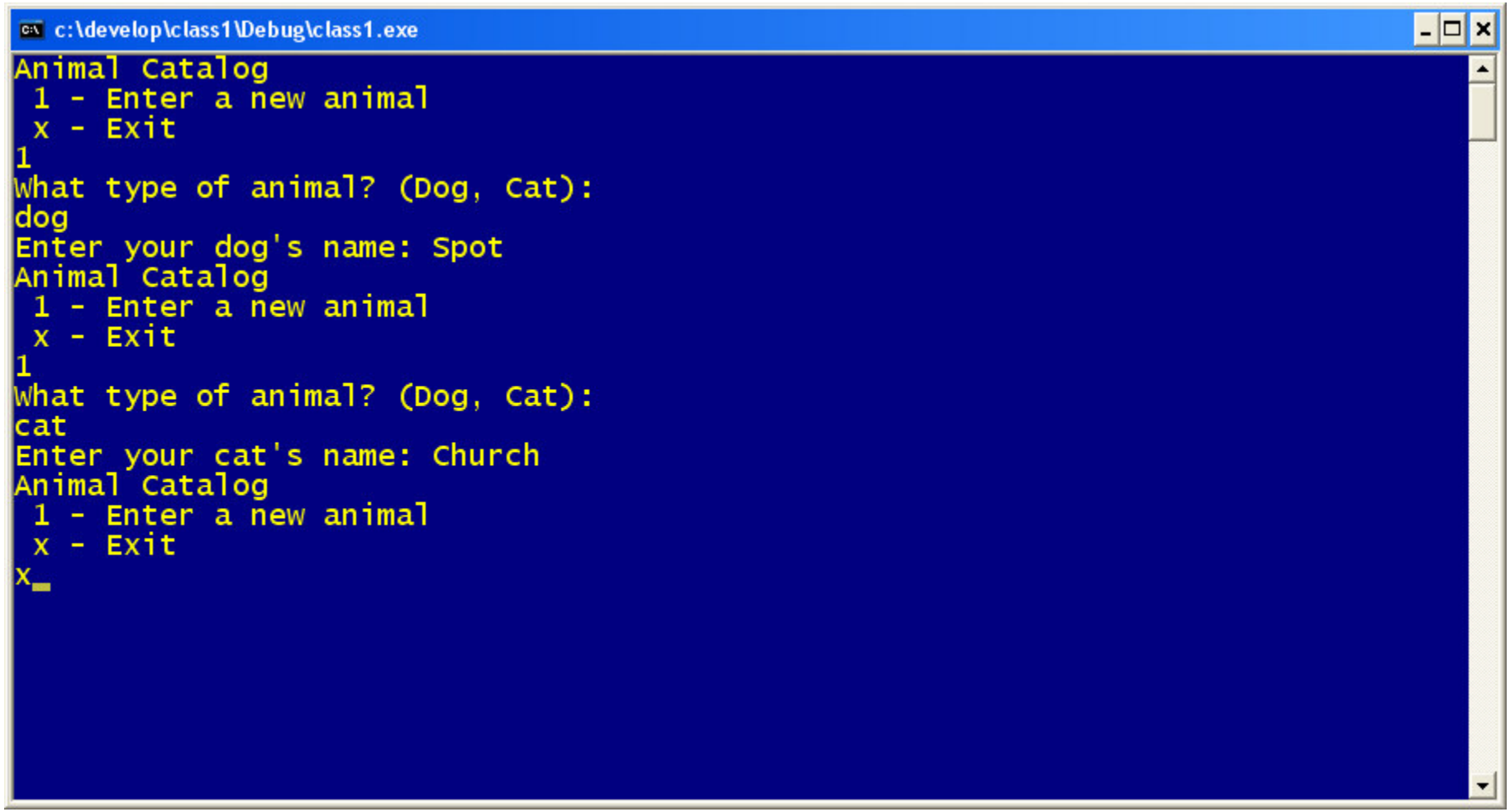
```
void main()
{
  // let's create a menu and put it in a loop
  while( true )
  {
    string input;
    cout << "Animal Catalog" << endl;
    cout << " 1 - Enter a new animal" << endl;
    cout << " x - Exit" << endl;
    cin >> input;

    switch( input[0] )
    {
    case '1':
      getAnimal();
      break;
    case 'x':
      // exit
      exit(0);
    default:
      cout << "I don't understand\n" << endl;
    }
  }
}
```

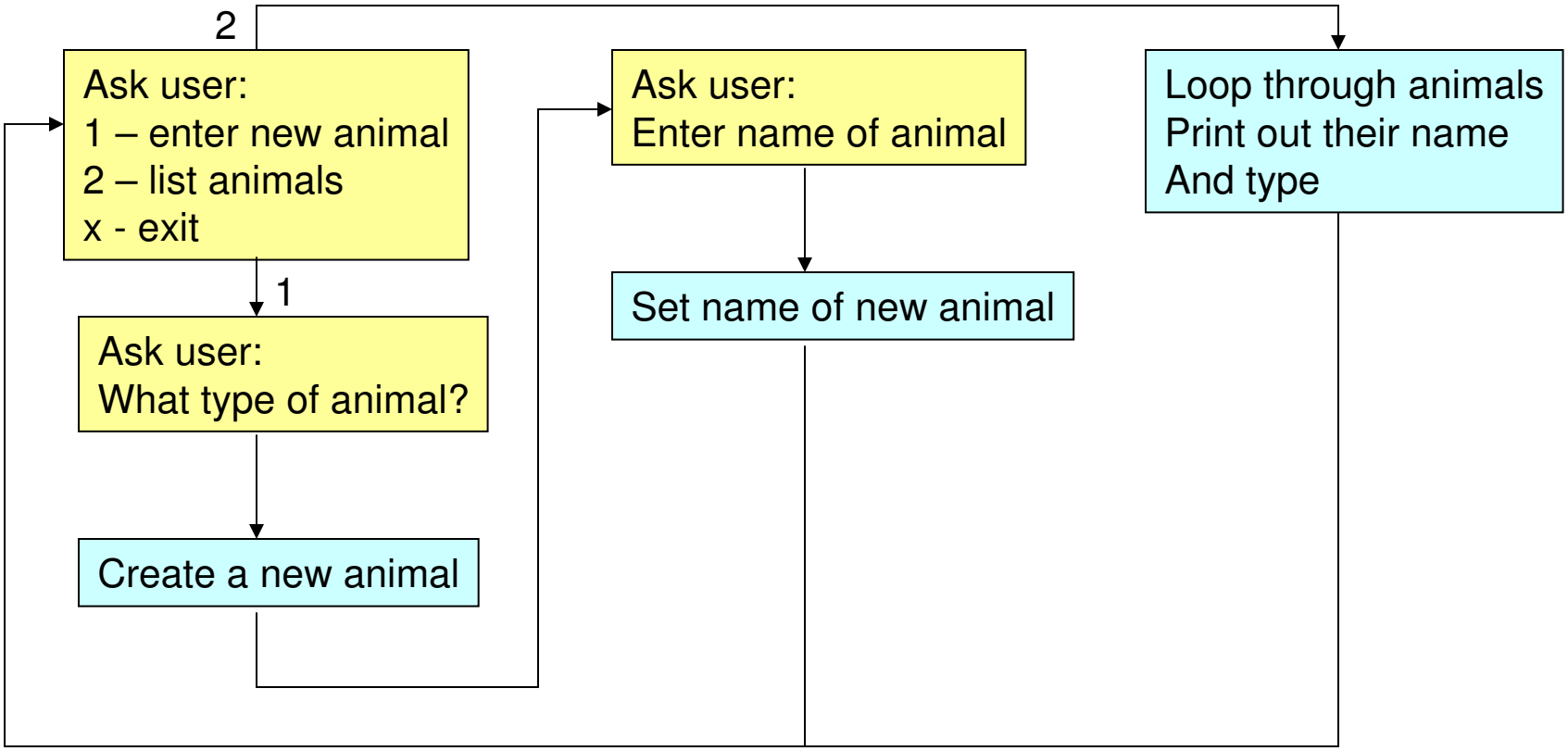
Creating our object

```
bool getAnimal()
{
  string input;
  cout << "What type of animal? (Dog, Cat):" << endl;
  cin >> input;
  if ( strcmpi( input.c_str(), "dog" ) == 0 ) // dog
  {
    animals.push_back( new Dog() );
  }
  else if ( strcmpi( input.c_str(), "cat" ) == 0 ) // cat
  {
    animals.push_back( new Cat() );
  }
  else
  {
    cout << "What?" << endl;
    return false;
  }
  cout << "Enter your " << input << "'s name: ";
  cin >> input;
  animals[ animals.size() - 1 ]->setName( input );
}
```

Our program


c:\develop\class1\Debug\class1.exe
Animal Catalog
1 - Enter a new animal
x - Exit
1
What type of animal? (Dog, Cat):
dog
Enter your dog's name: Spot
Animal Catalog
1 - Enter a new animal
x - Exit
1
What type of animal? (Dog, Cat):
cat
Enter your cat's name: Church
Animal Catalog
1 - Enter a new animal
x - Exit
x

Let's do a little more…



Our menu

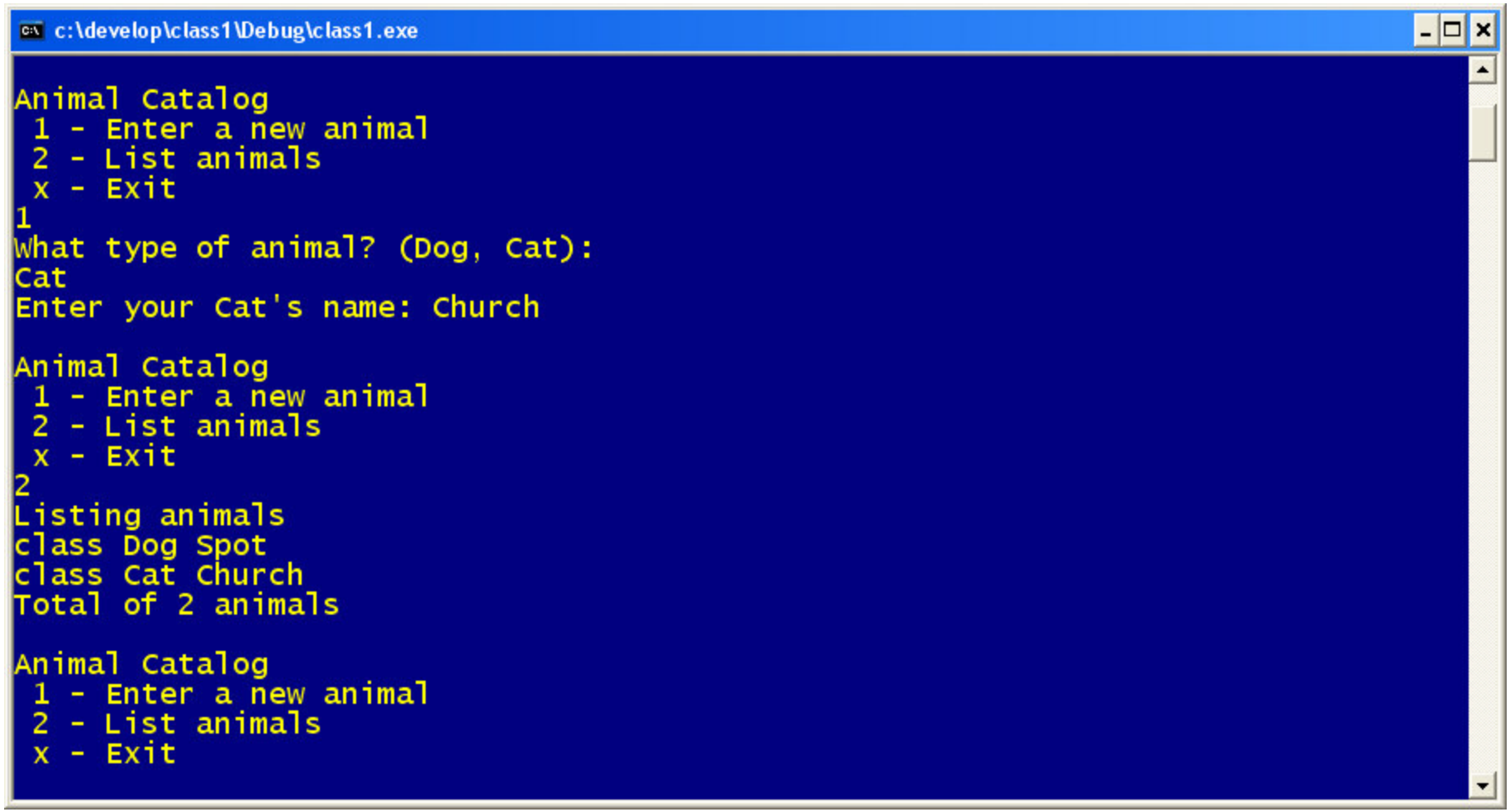
```
void main()
{
  // let's create a menu and put it in a loop
  while( true )
  {
    string input;
    cout << "\nAnimal Catalog" << endl;
    cout << " 1 - Enter a new animal" << endl;
    cout << " 2 - List animals" << endl;
    cout << " x - Exit" << endl;
    cin >> input;

    switch( input[0] )
    {
    case '1':
      getAnimal();
      break;
    case '2':
      listAnimals();
      break;
    case 'x':
      // exit
      exit(0);
    default:
      cout << "I don't understand\n" << endl;
    }
  }
}
```

Listing our objects

```
void listAnimals()
{
  cout << "Listing animals" << endl;
  for( int i = 0; i < animals.size(); i++ )
  {
    cout << typeid( *animals[i] ).name() << " ";
    cout << animals[i]->getName() << endl;
  }
  cout << "Total of " << animals.size() << " animals" << endl;
}
```

Our program


c:\develop\class1\Debug\class1.exe
Animal Catalog
1 - Enter a new animal
2 - List animals
x - Exit
1
What type of animal? (Dog, Cat):
Cat
Enter your Cat's name: Church
Animal Catalog
1 - Enter a new animal
2 - List animals
x - Exit
2
Listing animals
class Dog Spot
class Cat Church
Total of 2 animals
Animal Catalog
1 - Enter a new animal
2 - List animals
x - Exit

Serializing your objects

We can create menus for user input, so our program is at least usable...

But, it would be nice if we could save the objects the user created.

And, it would be nice if we could load the objects we saved.

This is called object serialization.

Serializing your objects

XML is a great format to use for serializing objects, but for this example we're just going to use comma delimited files.

Our file will end up looking like:

Saving our objects

```
void saveAnimals()
{
  ofstream outputFile( "animals" );
  for( int i = 0; i < animals.size(); i++ )
  {
    outputFile << typeid( *animals[i] ).name() << ",";
    outputFile << animals[i]->getName() << endl;
  }
  outputFile.close();
}
```

Loading our objects

```
void loadAnimals()
{
  animals.clear(); //empty our list

  ifstream inputFile( "animals" );

  string className;
  string animalName;
  char buffer[ 128 ];
  while ( inputFile.getline( buffer, sizeof buffer, ',' ) )
  {
    className = buffer;
    inputFile.getline( buffer, sizeof buffer );
    animalName = buffer;
    if ( className.compare( "class Dog" ) == 0 )
    {
      animals.push_back( new Dog() );
    }
    else
    {
      animals.push_back( new Cat() );
    }
    animals[ animals.size() - 1 ]->setName( animalName );
  }

  inputFile.close();
}
```

Designing classes and objects

Designing programs is the topic of many, many books.

Techniques for class and object design often follow methodologies that include:

- Unified Design
- Agile Modeling

Both of these techniques use the same language

Designing classes and objects

These methodologies start from how to find out what is required by the program and often record this as a list of requirements and use cases.

The use cases are stories that describe how the program might be used.

Designing classes and objects

Use cases can be used re-formed to determine what types of objects might be needed in out program.

The UML language can be used to draw the use cases, class diagrams, state charts, sequence diagrams and other items needed to design our program.

Designing classes and objects

If you'd like to find out more about designing programs using these methodologies visit:

http://www.omg.org/

http://agilemanifesto.org/

Anybody have any questions?

Anything else?