



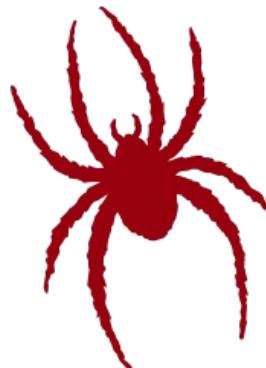
UNIVERSITY OF
RICHMOND

CMSC 240 Lecture 10

CMSC 240 Software Systems Development
Fall 2023

Today

- Inheritance
- Polymorphism
- Virtual functions
- Pure virtual functions and abstract classes



Today

- Inheritance
- Polymorphism
- Virtual functions
- Pure virtual functions and abstract classes



Inheritance

- Suppose you will define classes to model **cats**, **dogs**, and **birds**
 - These classes have many common features
 - What is the best way to design these classes to avoid redundancy?
-
- Object-oriented programming allows you to define new classes from existing classes
 - This is called **inheritance**

Superclasses and Subclasses

- Inheritance enables you to define a general class (i.e., a superclass) and later extend it to more specialized classes (i.e., subclasses)
- A subclass **inherits** from a superclass
 - For example, both a dog and a cat are animals
 - **Animal** is a superclass
 - **Dog** is a subclass of **Animal**
 - **Cat** is a subclass of **Animal**
 - This is an example of an **is-a** relationship
 - Dog **is-a** Animal
 - Cat **is-a** Animal

Superclass

Animal
- name: string
- age: int
- favoriteFood : string
+ Animal(name: string, age: int)
+ eat()
+ setFavoriteFood(food: string)
+ speak()
- sleep()

“is a”

“is a”

Subclasses

Dog
- height: float
- dogBreed: string
+ Dog(name: string, age: int, height: float)
+ setDogBreed(breed: string)
+ speak()

Cat
- whiskerLength: float
- numberOfLives: int
+ Cat(name: string, age: int, whiskerLength: float)
+ setNumberOfLives(num: int)
+ speak()

Superclasses and Subclasses

- A subclass inherits accessible data fields and methods from its superclass and may also add new data fields and methods
 - A subclass is not a subset of its superclass
 - A subclass usually contains more information and methods
- For example
 - `Animal` has a name, age, and favorite food
 - `Cat` also has whisker length, and number of lives
 - `Dog` also has height, and a dog breed

Superclasses and Subclasses

- A **superclass** is also called a “parent class” or “base class”
- A **subclass** is also called a “child class” or “derived class”
- A child class inherits from a parent class
- A subclass extends a superclass
- A derived class derives from a base class

Superclasses and Subclasses

- Remember, a class defines a type
- A type defined by a subclass is called a *subtype*, and a type defined by its superclass is called a *supertype*
- For example
 - **Cat** is a subtype of **Animal**, and
 - **Animal** is a supertype of **Dog**

Inheritance

```
class BaseClass
{
    // ... code for the base class
};

class DerivedClass : accessSpecifier BaseClass
{
    // ... code for the derived class
};
```

**public
protected
private**



Access Control with Inheritance

Public Inheritance

- **public members** of the base class
 - become public members of the derived class
- **protected members** of the base class
 - become protected members of the derived class
- **private members** of the base class are
 - not accessible directly from the derived class

Access Control with Inheritance

Protected Inheritance

- **both public and protected members** of the base class
 - become protected members of the derived class
- **private members** of the base class are
 - not accessible directly from the derived class

Access Control with Inheritance

Private Inheritance

- **both public and protected members** of the base class
 - become private members of the derived class
- **private members** of the base class are
 - not accessible directly from the derived class

Constructor and Destructor in Inheritance

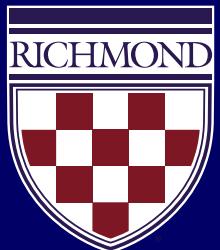
- When creating an object of the derived class, the **base class's constructor** is called **first**, followed by the derived class's constructor
- Conversely, when the object is destroyed, the **derived class's destructor** is called **first**, followed by the base class's destructor

```
1 #include <iostream>
2 using namespace std;
3
4 class Parent
5 {
6 public:
7     Parent()
8     {
9         cout << "1. Parent class under construction." << endl;
10    }
11 };
12
13 class Child : public Parent // Child inherits from the Parent
14 {
15 public:
16     Child()
17     {
18         cout << "2. Child class under construction." << endl;
19     }
20 };
21
22 int main()
23 {
24     // Create a new instance of the child class.
25     Child childInstance;
26 }
```

```
1 #include <iostream>
2 using namespace std;
3
4 class Parent
5 {
6 public:
7     Parent()
8     {
9         cout << "1. Parent class under construction." << endl;
10    }
11 }
12
13 class Child : public Parent // Child inherits from the Parent
14 {
15 public:
16     Child()
17     {
18         cout << "2. Child class under construction." << endl;
19     }
20 }
21
22 int main()
23 {
24     // Create a new instance of the child class.
25     Child childInstance;
26 }
```

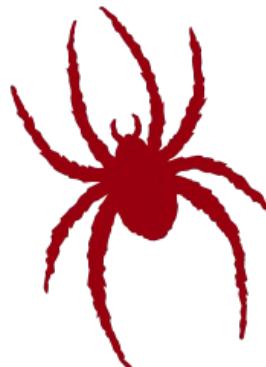
1. Parent class under construction.
2. Child class under construction.

Ask a question



Today

- Inheritance
- Polymorphism
- Virtual functions
- Pure virtual functions and abstract classes



Polymorphism

- Polymorphism is a foundational concept in object-oriented programming that enables objects of different classes to be treated as objects of a common super class
- The term "**polymorphism**" is derived from Greek and means "**having multiple forms**"
- At its core, polymorphism allows one interface to represent **many different types** of objects or methods

Polymorphism

- Remember, a class defines a type
- A type defined by a subclass is called a subtype, and a type defined by its superclass is called a supertype
- For example
 - Dog is a subtype of **Animal**, and
 - **Animal** is a supertype for **Cat**
- Polymorphism means that a variable of a supertype can refer to a subtype object
 - For example, an **Animal** could be used to refer to a **Cat** or **Dog**

Polymorphism

- An object of a subtype can be used wherever its supertype value is required

For example: the **animals** vector is a list of pointers to **Animal** types. But we load it with **Dog** and **Cat** types.

```
// Create a dog and a cat.  
Dog woofer{"Woofer", 3, 36.4};  
Cat cheddar{"Cheddar", 5, 3.1};  
  
// Create a vector of animal pointers.  
vector<Animal*> animals;  
  
// Add addresses to a dog and a cat.  
animals.push_back(&woofer);  
animals.push_back(&cheddar);
```

Polymorphism

- An object of a subtype can be used wherever its supertype value is required

Actual types

Declared type

```
// Create a dog and a cat.  
Dog woofer{"Woofer", 3, 36.4};  
Cat cheddar{"Cheddar", 5, 3.1};  
  
// Create a vector of animal pointers.  
vector<Animal*> animals;  
  
// Add addresses to a dog and a cat.  
animals.push_back(&woofer);  
animals.push_back(&cheddar);
```

Today

- Inheritance
- Polymorphism
- Virtual functions
- Pure virtual functions and abstract classes



Virtual Functions

- The **virtual** keyword plays a crucial role in enabling polymorphic behavior
 - When a function is declared as **virtual** in a base class, it indicates that this function can be overridden by a derived class
 - When a pointer to the base class type points to an object of a derived class, a call to a virtual function will invoke the most derived version of that function for the actual object being pointed to
- With the **virtual** keyword, the function call is dynamically bound to the appropriate version at runtime

```
1 #ifndef ANIMAL_H
2 #define ANIMAL_H
3 #include <string>
4
5 class Animal
6 {
7 public:
8     Animal(std::string name, int age);
9     void eat();
10    void setFavoriteFood(std::string favorite);
11    virtual void speak(); 
12 private:
13     std::string name;
14     int age;
15     std::string favoriteFood;
16     void sleep();
17 };
18
19#endif
```

Today

- ~~Inheritance~~
- ~~Polymorphism~~
- ~~Virtual functions~~
- Pure virtual functions and abstract classes



Pure Virtual Function

pure virtual function

- a virtual function with an = 0 assignment
- indicating that there is no implementation for that function
- any concrete derived class must provide an implementation

```
5  class Animal
6  {
7  public:
8      Animal(std::string name, int age);
9      void eat();
10     void setFavoriteFood(std::string favorite);
11     virtual void speak() = 0;
```

Abstract Class

- An **abstract class** is a class that either defines or inherits at least one function for that is pure virtual
- **You can not create an instance of an abstract class**

```
5  class Animal
6  {
7  public:
8      Animal(std::string name, int age);
9      void eat();
10     void setFavoriteFood(std::string favorite);
11     virtual void speak() = 0;
```



Ask a question

