Inheritance

CSCI 201
Principles of Software Development

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu
Outline

• Inheritance
Inheritance

- Inheritance is a form of software reuse in which you create a class that absorbs an existing class’s data and behaviors and enhances them with new capabilities.

- When creating a class, you can designate that the new class should inherit the members of an existing class:
  - The existing class is called the base class (or parent class).
  - The new class is called the derived class (or child class).
Single vs Multiple Inheritance

- Single inheritance allows a derived class to inherit from only one base class
  - Java supports single inheritance

- Multiple inheritance allows a derived class to inherit from more than one base class
  - C++ supports multiple inheritance
  - There is a potential problem with multiple inheritance if more than one base class provides an implementation for the same function
# Multiple Inheritance in C++

```cpp
#include <iostream>
using namespace std;

class Email_Reader {
    public:
    void read_email() {
        cout << "reading email" << endl;
    }
};

class Telephone {
    public:
    void make_call() {
        cout << "making call" << endl;
    }
};

class IPhone : public Telephone, public Email_Reader {
    public:
    void buy_app() {
        cout << "buying app" << endl;
    }
};

void main() {
    IPhone ip;
    ip.buy_app();
    ip.make_call();
    ip.read_email();
}
```
Multiple Inheritance in C++

```cpp
#include <iostream>
using namespace std;

class Email_Reader {
  public:
    void read_email() {
      cout << "reading email" << endl;
    }
    void send_email() {
      cout << "Email sending email";
    }
};

class Telephone {
  public:
    void make_call() {
      cout << "making call" << endl;
    }
    void send_email() {
      cout << "Telephone sending email";
    }
};

class IPhone : public Telephone, public Email_Reader {
  public:
    void buy_app() {
      cout << "buying app" << endl;
    }
};

void main() {
  IPhone ip;
  ip.buy_app();
  ip.make_call();
  ip.read_email();
  // does this line compile?
  ip.send_email();
}
```
# Multiple Inheritance in C++

```cpp
#include <iostream>
using namespace std;

class Email_Reader {
  public:
    void read_email() {
      cout << "reading email" << endl;
    }
    void send_email() {
      cout << "Email sending email";
    }
};

class Telephone {
  public:
    void make_call() {
      cout << "making call" << endl;
    }
    void send_email() {
      cout << "Telephone sending email";
    }
};

class IPhone : public Telephone, public Email_Reader {
  public:
    void buy_app() {
      cout << "buying app" << endl;
    }
};

void main() {
  IPhone ip;
  ip.buy_app();
  ip.make_call();
  ip.read_email();
  ip.Telephone::send_email();
  ip.Email_Reader::send_email();
}
```
Inheritance vs Composition

▪ is-a Relationship
  › If an object has an “is-a” relationship with another object, inheritance will be used
  › Vehicle, Car, Truck, Motorcycle

▪ has-a Relationship
  › If an object has a “has-a” relationship with another object, composition will be used
  › Car, Steering Wheel, Brake Pedal, Speedometer
Access Methods

- **public**
  - Any other class has access to public member variables and methods

- **protected**
  - Subclasses and classes within the same package have access to protected member variables and methods

- **<package>**
  - Other classes within the same package have access to member variables and methods (which is the default access)

- **private**
  - Only the current class has access to the member variables and methods
**Instantiating a Child Class**

- To inherit from another class, use the keyword `extends` immediately following the name of the class, followed by the name of the class from which you would like to inherit.
- When a child class is instantiated, the parent class must be instantiated **first** in the child class’s constructor.
  - This will happen automatically by the compiler calling the parent class’s default constructor unless we explicitly instantiate the parent.
  - Note that if there is no default constructor in the parent, we MUST explicitly call the parent class’s constructor from the child.
- When we call the parent class’s constructor from the child, it must be the first line of code in the child class’s constructor.

```java
1 public class Shape {
2   protected char name;
3   public Shape(char n) {
4     name = n;
5   }
6 }
```

```java
1 public class TwoDShape extends Shape {
2   public TwoDShape(char name) {
3     super(name);
4   }
5 }
```
Inheritance Example

```java
1  class Parent {
2      private int num;
3      public Parent(int num) {
4          this.num = num;
5      }
6      public int meth() {
7          return num;
8      }
9  }
10
11  public class Child extends Parent {
12      public Child() {
13      }
14      public static void main(String [] args) {
15          Child c = new Child();
16          System.out.println(c.meth());
17      }
18  }
```
Inheritance Example

```java
1  class Parent {
2      private int num;
3      public Parent(int num) {
4          this.num = num;
5      }
6      public int meth() {
7          return num;
8      }
9  }
10
11  public class Child extends Parent {
12      public Child() {
13          super(10);
14      }
15      public static void main(String[] args) {
16          Child c = new Child();
17          System.out.println(c.meth());
18      }
19  }
```
Inheritance Hierarchy

- To show a child class and a parent class in a diagram, we draw a line connecting the child class to the parent class where the parent class is above the child.
- Assume the following hierarchy for the rest of this lecture.

```
Shape

<table>
<thead>
<tr>
<th>2-D Shape</th>
<th>3-D Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangle</td>
<td>Ellipsoid</td>
</tr>
<tr>
<td>Rectangle</td>
<td>Cone</td>
</tr>
<tr>
<td>Ellipse</td>
<td>Box</td>
</tr>
<tr>
<td>Square</td>
<td>Sphere</td>
</tr>
<tr>
<td>Circle</td>
<td>Cube</td>
</tr>
</tbody>
</table>
```
Inheritance Example

```java
1  class Shape {
2      protected char name;
3  public Shape(char n) {
4      name = n;
5  }
6  public void printName() {
7      System.out.println(name);
8  }
9  }
```

```java
1  class TwoD extends Shape {
2      public TwoD(char name) {
3          super(name);
4      }
5  }
```

```java
1  class Triangle extends TwoD {
2      private float base;
3      private float height;
4  public Triangle(char nm, float b, float h) {
5          super(nm);
6          base = b;
7          height = h;
8      }
9  public float getArea() {
10         return 0.5f * base * height;
11  }
12  }
```

```java
1  class Rectangle extends TwoD {
2      protected float width, length;
3  public Rectangle(char nm, float w, float l) {
4          super(nm);
5          width = w;
6          length = l;
7      }
8  public float getArea() {
9          return width * length;
10     }
11  }
```

```java
1  class Square extends Rectangle {
2      public Square(char nm, float s) {
3          super(nm, s, s);
4      }
5  }
```
Inheritance Example (cont.)

```java
1   public class Test {
2       public static void main() {
3           Triangle t = new Triangle(‘t’, 3.0f, 4.0f);
4           t.printName();
5           System.out.println(t.getArea());
6           Rectangle r = new Rectangle(‘r’, 5.0f, 6.0f);
7           r.printName();
8           System.out.println(r.getArea());
9           Square sq = new Square(‘s’, 3);
10          sq.printName();
11          System.out.println(sq.getArea());
12       }
```