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
  - What if more than one parent implements 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++

```
1 #include <iostream>
2 using namespace std;
3
4 class Email_Reader {
5   public:
6     void read_email() {
7       cout << "reading email" << endl;
8     }
9     void send_email() {
10       cout << "Email sending email";
11     }
12 }
13
14 class Telephone {
15   public:
16     void make_call() {
17       cout << "making call" << endl;
18     }
19     void send_email() {
20       cout << "Telephone sending email";
21     }
22 }
23
24 class IPhone : public Telephone, public Email_Reader {
25   public:
26     void buy_app() {
27       cout << "buying app" << endl;
28     }
29 }
30
31 void main() {
32   IPhone ip;
33   ip.buy_app();
34   ip.make_call();
35   ip.read_email();
36   ip.send_email(); // does this line compile?
37 }
```
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();
}
```

- **Telephone**
  - `make_call()`
  - `send_email()`

- **Email_Reader**
  - `read_email()`
  - `send_email()`

- **IPhone**
  - `buy_app()`
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 (which is the default access)

- **private**
  - Only the current class has access to the member variables and methods

What can I eat?
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
public class Shape {
    protected char name;
    public Shape(char n) {
        name = n;
    }
}

public class TwoDShape extends Shape {
    public TwoDShape(char name) {
        super(name);
    }
}
```
```java
Inheritance Example

```class Parent {
    private int num;
    public Parent(int num) {
        this.num = num;
    }
    public int meth() {
        return num;
    }
}

public class Child extends Parent {
    public Child() {
    }
    public static void main(String [] args) {
        Child c = new Child();
        System.out.println(c.meth());
    }
}
Inheritance Solution #1

```java
class Parent {
    private int num;
    public Parent(int num) {
        this.num = num;
    }
    public int meth() {
        return num;
    }
}

public class Child extends Parent {
    public Child() {
        super(10);
    }
    public static void main(String[] args) {
        Child c = new Child();
        System.out.println(c.meth());
    }
}
```
Inheritance Solution #2

```java
class Parent {
    private int num;

    public Parent() {
    }

    public Parent(int num) {
        this.num = num;
    }

    public int meth() {
        return num;
    }
}

public class Child extends Parent {
    public Child() {
    }

    public static void main(String[] args) {
        Child c = new Child();
        System.out.println(c.meth());
    }
}
```
```java
public class Child extends Parent {

    public Child() {

    }

    public static void main(String [] args) {
        Child c = new Child();
        System.out.println(c.meth());
    }
}
```
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

2-D Shape
- Triangle
- Rectangle
- Ellipse
- Square
- Circle

3-D Shape
- Ellipsoid
- Cone
- Box
- Cube
```
Inheritance Example

class Shape {
  protected char name;
  public Shape(char n) {
    name = n;
  }
  public void printName() {
    System.out.println(name);
  }
}

class TwoD extends Shape {
  public TwoD(char name) {
    super(name);
  }
}

class Triangle extends TwoD {
  private float base;
  private float height;
  public Triangle(char nm, float b, float h) {
    super(nm);
    base = b;
    height = h;
  }
  public float getArea() {
    return 0.5f * base * height;
  }
}

class Rectangle extends TwoD {
  protected float width, length;
  public Rectangle(char nm, float w, float l) {
    super(nm);
    width = w;
    length = l;
  }
  public float getArea() {
    return width * length;
  }
}

class Square extends Rectangle {
  public Square(char nm, float s) {
    super(nm, s, s);
  }
}
Inheritance Example (cont.)

```java
public class Test {
    public static void main() {
        Triangle t = new Triangle('t', 3.0f, 4.0f);
        t.printName();
        System.out.println(t.getArea());
        Rectangle r = new Rectangle('r', 5.0f, 6.0f);
        r.printName();
        System.out.println(r.getArea());
        Square sq = new Square('s', 3);
        sq.printName();
        System.out.println(sq.getArea());
    }
```