Abstract Classes
Interfaces

CSCI 201
Principles of Software Development

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

• Abstract Classes
• Interfaces
Abstract Classes

- An abstract class is a way for parent classes to guarantee that child classes provide an implementation for a specific method
  - Consider the Shape example. Even though a Shape does not know how to find the area of a Triangle or Rectangle, it could require that both of those classes implement a getArea() method

- Abstract methods only contain declarations but no implementations
  - Any class that contains an abstract method must be declared abstract

- Abstract classes cannot be instantiated since not all of the methods have implementations

- Any class that inherits from an abstract class must implement all of the abstract methods or declare itself abstract
  - When a class implements an abstract method, it is said to override that method
Abstract Class Example

```java
1 abstract class Parent {
2    public abstract int meth1();
3    public int meth2() {
4        return 10;
5    }
6 }
7
8 class Child extends Parent {
9    public int meth1() {
10       return 20;
11    }
12 }
13
14 public class Test {
15    public static void main(String [] args) {
16       Child c = new Child();
17       System.out.println(c.meth1());
18       System.out.println(c.meth2());
19       Parent p = new Parent();
20       System.out.println(p.meth1());
21       System.out.println(p.meth2());
22    }
23 }
```
abstract class Parent {
    public abstract int meth1();
    public int meth2() {
        return 10;
    }
}

class Child extends Parent {
    public int meth1() {
        return 20;
    }
}

class Test {
    public static void main(String[] args) {
        Child c = new Child();
        System.out.println(c.meth1());
        System.out.println(c.meth2());
        Parent p = new Child();
        System.out.println(p.meth1());
        System.out.println(p.meth2());
    }
}
Abstract Class Example

```java
1 abstract class Parent {
2    public abstract int meth1();
3    public int meth2() {
4        return 10;
5    }
6 }
7
8 class Child extends Parent {
9    public int meth1() {
10       return 20;
11    }
12 }
13
14 public class Test {
15    public void foo(Parent p) {
16       System.out.println(p.meth1());
17       System.out.println(p.meth2());
18    }
19    public static void main(String [] args) {
20       Child c = new Child();
21       System.out.println(c.meth1());
22       System.out.println(c.meth2());
23       foo(c);
24    }
```
Outline

• Abstract Classes
• Interfaces
Interfaces

- An interface is similar to a class, but there are no method implementations in it (not even inherited)

- When a class **implements** an interface, it must implement all of the methods in the interface
  - If it doesn’t implement all of the methods, it has then inherited an abstract method, so the class must be declared abstract

- A class can implement as many interfaces as it wants
  - This is how Java deals with supporting something like multiple inheritance
  - This is different than multiple inheritance though. How?
    - If the same method is inherited from more than one interface, there is no implementation, so there is no confusion

- If interfaces inherit from other interfaces, they will **extend** them
Multiple Inheritance and Interfaces

- In C++, we had a problem with multiple inheritance when the same function was implemented in two different branches of the hierarchy.

Program:
```cpp
Telephone
  make_call()
  send_email()

Email_Reader
  read_email()
  send_email()

IPhone
  buy_app()

Which send_email() function gets called when you call it on an instance of IPhone?

IPhone ip;
ip.send_email();
```

It doesn’t matter which one gets called if the Telephone and Email_Reader classes only contained function definitions and the only implementation was in IPhone.
Interface Example

```java
1  interface Parent {
2    public int meth1();
3  }

4
5  abstract class Child implements Parent {
6    public abstract int meth2();
7    public int meth1() {
8      return 20;
9    }
10 }

11 class GrandChild extends Child {
12   public int meth2() {
13     return 30;
14   }
15 }
16
17
18 public class Test {
19   public static void main(String [] args) {
20     GrandChild gc = new GrandChild();
21     System.out.println(gc.meth1());
22     System.out.println(gc.meth2());
23     Child c = new Child();
24     System.out.println(c.meth1());
25     System.out.println(c.meth2());
26   }
27 }
```
interface Parent {
  public int meth1();
}

abstract class Child implements Parent {
  public abstract int meth2();
  public int meth1() {
    return 20;
  }
}

class GrandChild extends Child {
  public int meth2() {
    return 30;
  }
}

public class Test {
  public void foo(Child c) {
    System.out.println(c.meth1());
    System.out.println(c.meth2());
  }
  public static void main(String [] args) {
    GrandChild gc = new GrandChild();
    System.out.println(gc.meth1());
    System.out.println(gc.meth2());
    foo(gc);
  }
}
### Interface Example

1 interface Parent {
2    public int meth1();
3 }  
4
5 abstract class Child implements Parent {
6    public abstract int meth2();
7    public int meth1() {
8     return 20;
9   }
10 }
11
12 class GrandChild extends Child {
13   public int meth2() {
14     return 30;
15   }
16 }
17
18 public class Test {
19   public void foo(Parent p) {
20     System.out.println(p.meth1());
21     System.out.println(p.meth2());
22   }
23   public static void main(String [] args) {
24     GrandChild gc = new GrandChild();
25     System.out.println(gc.meth1());
26     System.out.println(gc.meth2());
27     foo(gc);
28   }
29 }

This code doesn’t compile because `meth2()` can’t be called on a Parent object on line 21.