

CS102

Programming II

CLASS (PART - III)

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

TOPICS

- Abstract classes
- Interfaces

CS102

Programming II

ABSTRACT CLASSES

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

ABSTRACT CLASSES

- An **abstract** class in Java is a restricted class that cannot be instantiated on its own (you cannot create objects)
- We use **abstract** class as a template (base class), other classes can **extend** from this abstract class. The extended classes are considered as **concrete** classes.
- **abstract** method is declared without a body (no { }).
- Subclasses must override and implement all abstract methods unless the subclass is also abstract.

ABSTRACT CLASSES AND METHODS

```
abstract class Phone {  
    String model;  
    Phone(String model) {  
        this.model = model;  
    }  
    //no code for abstract method:  
    abstract void powerOn();  
    public String getModel() {  
        return model;  
    }  
}
```

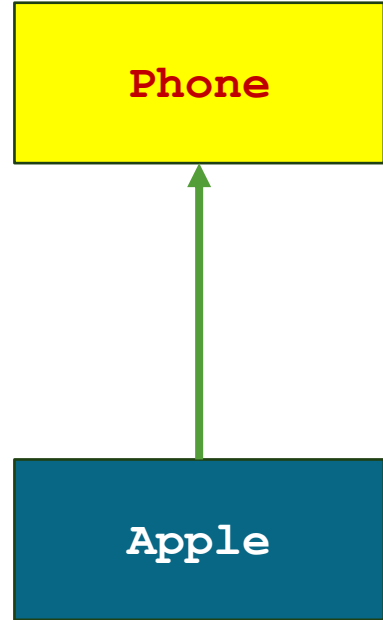
```
public class Apple extends Phone {  
    Apple(String model) {  
        super(model); //abstract class  
    }  
    @Override  
    public void powerOn() {  
        System.out.println(getModel() +  
            ": Display logo");  
    }  
}
```

ABSTRACT CLASSES AND METHODS

- Make a Phone object using super constructor
- Call a @Override method using abstract class

```
public static void main(String [] args){  
    Phone A = new Apple("iPhone 17");  
    A.powerOn() ;  
}
```

Base / Parent class



Child / sub class

- Phone class is **abstract**. Phone object A is instantiated to Apple (Polymorphism).
- **abstract** classes cannot instantiate their own object; **Phone A = new Phone()** is **not allowed**

ABSTRACT CLASSES AND METHODS

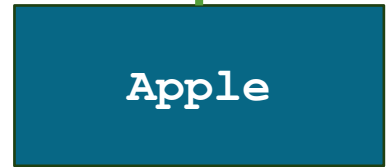
- Make a Phone object using super constructor
- Call a @Override method using abstract class

```
public static void main(String [] args){  
    Phone A = new Apple("iPhone 17");  
    A.powerOn() ;  
}
```

Base / Parent class



Child / sub class



- Method `powerOn()` is **abstract**; it defines what method should do (e.g. defines parameters and return type; but not how to do it! (no code). The **extending** class will define code for this method using `@override`.

ABSTRACT CLASSES AND METHODS

Feature	Normal class	Abstract class
Create objects directly	Yes	No
Can have attributes	Yes	Yes
Can have constructors	Yes	Yes
Can have normal methods	Yes	Yes
Can have abstract methods	No	Yes
Can be inherited	Yes	Yes

Can we create abstract attributes/variables in a abstract class? -> NO

ABSTRACT CLASSES AND METHODS

- An abstract class does not mean every method must be **abstract**. A class can contain both **abstract** and concrete methods.

```
abstract class Phone {  
    String model;  
    Phone(String model) {  
        this.model = model;  
    }  
    abstract void powerOn();  
    public String getModel() {  
        return model;  
    }  
}
```

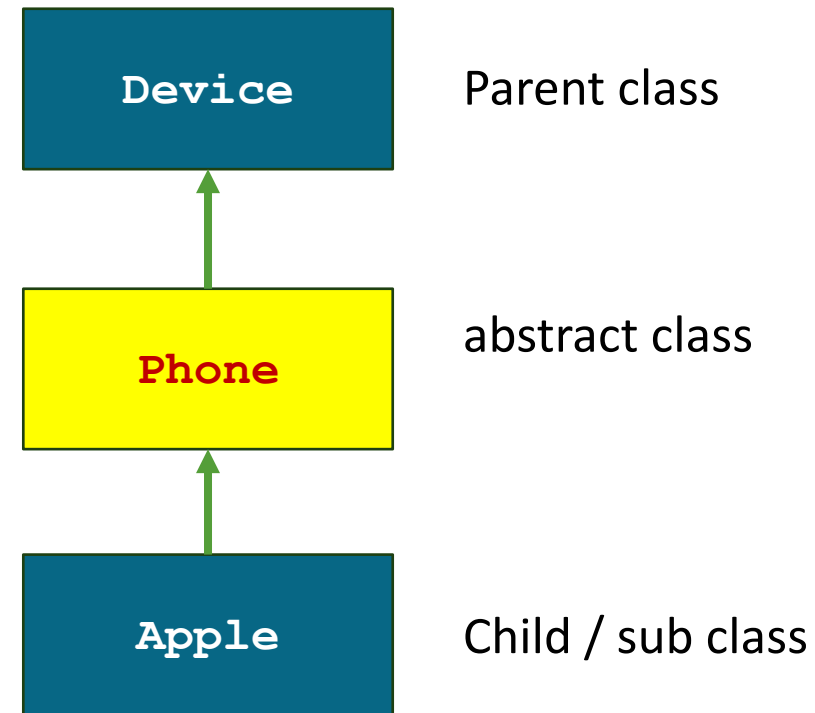
attributes/variables cannot be **abstract**

abstract method has no code,
only definition/header

concrete method has code with,
definition/header

ABSTRACT CLASSES AND METHODS

- A class that contains **abstract** methods must be **abstract**
- A subclass can be **abstract** even if its superclass is concrete
- A subclass can override a method from its superclass to define it abstract.
- *This is rare*



CS102

Programming II

INTERFACE

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

INTERFACE

- An **interface** is a blueprint of a class that defines **what** a class can do without telling **how** it does it.
- An **interface** is a strict contract that contains only constants and **abstract** methods
- Any class that implements an interface promises to provide concrete code for all of its declared methods.
- Similar to **abstract class** BUT the intention is to specify common behavior for objects
- All data fields are **public final static** and often used as constants
- All methods are **public abstract** in an interface. No code is provided for methods

INTERFACE

```
interface Phone {  
    //variables:public static final  
    int MAX_CHARGE=95;  
  
    //no code for method charge()  
    void charge();  
}
```

```
public class Apple implements Phone {  
    String name;  
    Apple(String name){  
        this.name = name;  
    }  
    @Override  
    public void charge(){  
        System.out.println("Charge: " + MAX_CHARGE);  
    }  
}
```

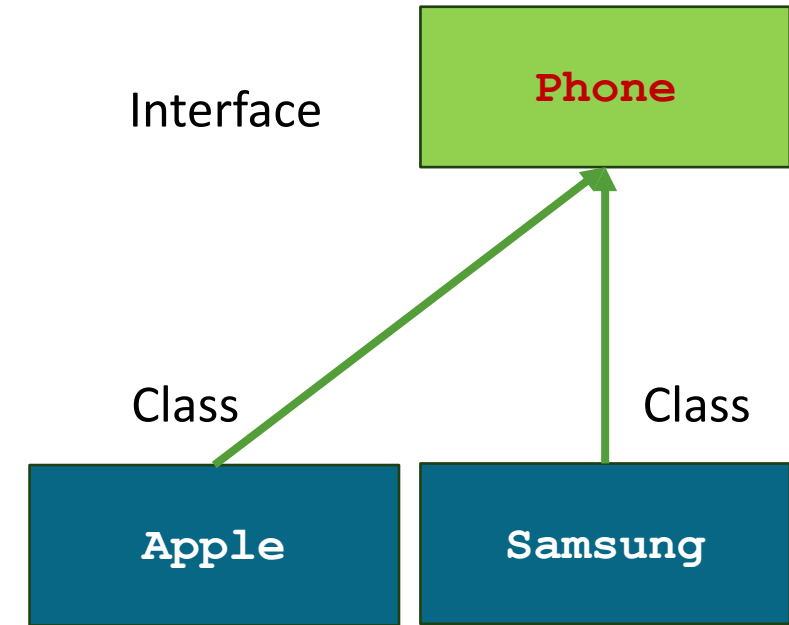
```
public class Samsung implements Phone {  
    String name;  
    Samsung(String name){  
        this.name = name;  
    }  
    @Override  
    public void charge(){  
        System.out.println("Charge: " + MAX_CHARGE);  
    }  
}
```

INTERFACE

- Make two Phone objects using **interface**
- Call a @Override method

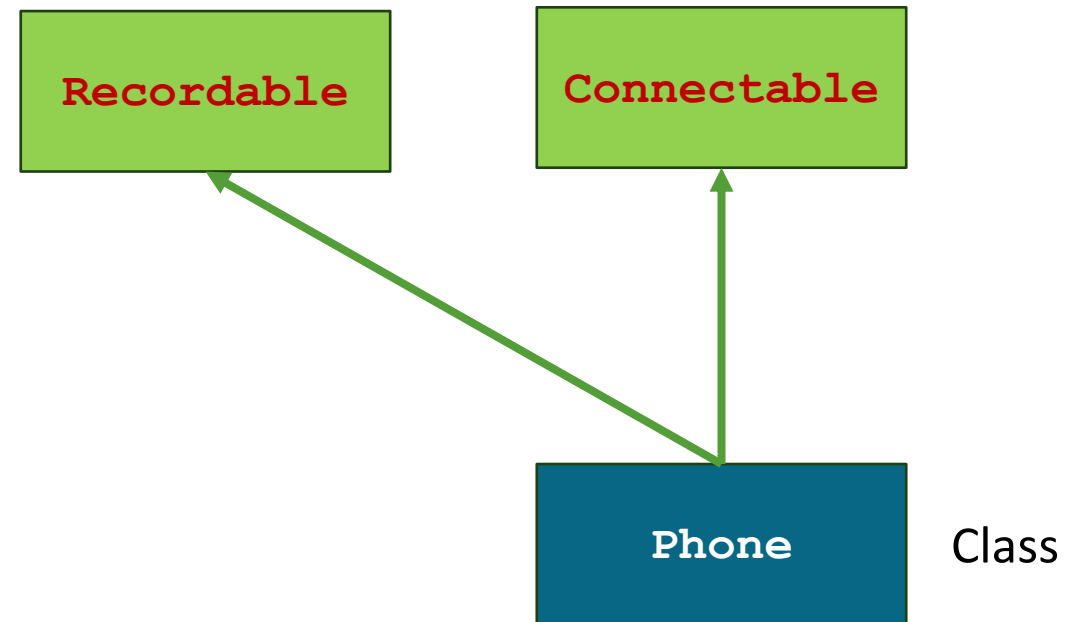
```
public static void main(String [] args){  
    Apple B = new Apple("iPhone 17");  
    Samsung C = new Samsung("Galaxy S25");  
    B.charge() ;  
    C.charge() ;  
}
```

- Phone is a **interface**. Apple object B is instantiated to Apple.
- **interface** cannot instantiate their own object;
Phone C = new Phone() is **not allowed**



INTERFACE

- We can define multiple **interface**
- A class can implement many **interface**
- Java supports multiple-inheritance through interface



INTERFACE

```
interface Recordable {  
  
    //no code for method charge()  
    void startRecording();  
}
```

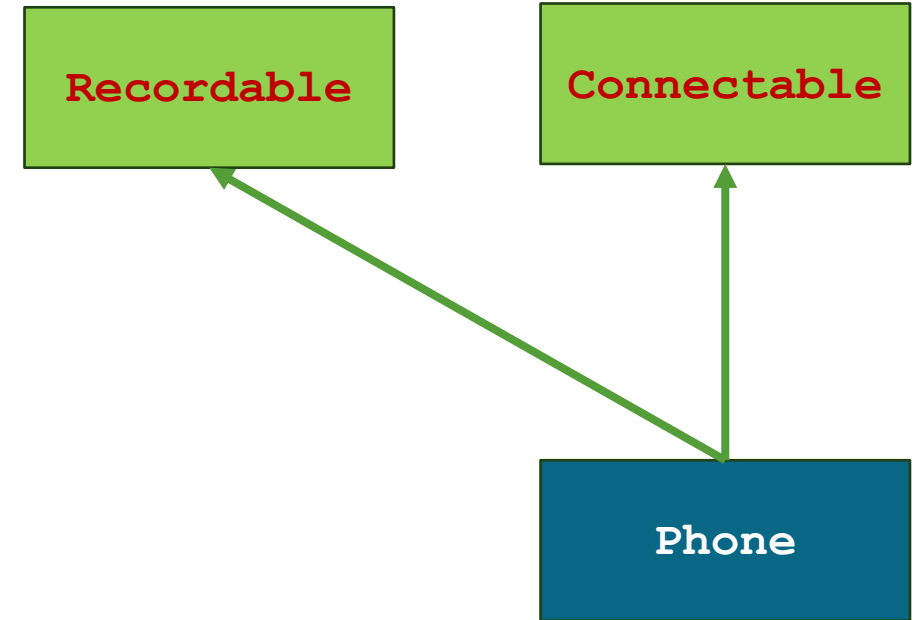
```
interface Connectable {  
  
    //no code for method charge()  
    void WiFiConnect();  
}
```

```
public class Apple implements Recordable, Connectable {  
    String name;  
    Apple(String name){  
        this.name = name;  
    }  
    @Override  
    public void startRecording(){  
        System.out.println(name+ ": started recording");  
    }  
    @Override  
    public void WiFiConnect(){  
        System.out.println(name+ ": connected using WiFi");  
    }  
}
```

INTERFACE

- Make a Phone object
- Call a @Override method defined in interface

```
public static void main(String [] args){  
    Apple D = new Apple("iPhone 17");  
    D.startRecording() ;  
    D.WiFiConnect() ;  
}
```



CS102

Programming II

COMPARING
ABSTRACT CLASS VS
INTERFACE

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

COMPARING ABSTRACT CLASS VS INTERFACE

Feature	Abstract class	Interface
Declaration	abstract class	interface
Can create objects?	No, directly	No, directly
Abstract methods	Yes	Yes
Concrete methods	Yes	only static
Instance attributes	Yes	No instance attributes
Fields	Any valid field modifiers	public static final implicitly
Constructors	Yes	No
Access modifiers	public, protected, private	Abstract methods are public
Keyword used by child	extends	implements
Inheritance	A class extends one class	A class can implement multiple interfaces
Can inherit an interface?	Yes, using implements	Yes, using extends