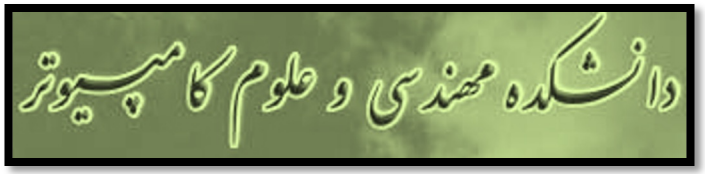


برنامه نویسی پیشرفته

وحیدی اصل

Abstract and Interface

مزایای شیئ گرایی- انتزاع (Abstraction)

- **پنهان کردن جزئیات داخلی از چشمان کاربر، انتزاع گفته می شود. کاربر، فقط باید کارکرد (قابلیتها) یا خروجی وسیله را منطبق بر نیازهای خویش بداند.**

- در جاوا، از کلاس abstract و interface برای رسیدن به انتزاع استفاده می کنیم.

تفاوت کپسوله بندی و انتزاع

- یک دوربین عکاسی دیجیتالی زیر را در نظر بگیرید:

- برروی راهنمای این دوربین نوشته شده:

"کاربر محترم، در هنگام استفاده از دوربین دیجیتالی کافیست برروی دکمه های zoom in و zoom out کلیک کنید و در هنگام زوم شدن دوربین حرکت لنز را حس خواهید نمود. "

- حال اگر دوربین را بازکنید، با مکانیزم پیچیده آن روبرو خواهید شد که برای شما قابل فهم نخواهد بود. فشردن دکمه عکاسی و گرفتن عکس (نتایج دلخواه) انتزاع نامیده می شود.

انتزاع در جاوا (abstraction)

- با Abstraction تنها موارد مهم و امکانات یک کلاس به کاربر نشان داده می‌شود و جزئیات داخلی (نحوه پیاده‌سازی zoom in یا zoom out) از کاربر پنهان می‌شود.
- انتزاع سبب می‌شود کاربر برروی قابلیتهای یک شیئ به جای نحوه پیاده‌سازی این قابلیتها متمرکز شود.

کلاس abstract

- کلاسی که با کلمه کلیدی abstract بیان شده باشد، کلاس abstract نامیده می‌شود.

```
abstract class A{}
```

- متدی که با کلمه abstract اعلان شده و بدنه آن خالی است (پیاده‌سازی ندارد) متد abstract نامیده می‌شود.

```
abstract void printStatus();//no body and abstract
```

کلاس abstract- مثال

- در این مثال، کلاس Bike یک کلاس abstract است که حاوی یک متد abstract به نام run() می‌باشد. پیاده‌سازی آن در کلاس فرزندش Honda ارایه شده است.

```
abstract class Bike{
  abstract void run();
}

class Honda4 extends Bike{
void run(){System.out.println("running safely..");}

public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Test it Now

```
running safely..
```

کلاس abstract- مثال دیگر

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}

class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}

//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();
s.draw();
}
}
```

Test it Now

```
drawing circle
```

کلاس abstract- مثال بانک

```
 abstract class Bank{
abstract int getRateOfInterest();
}

class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 6; }
}

class TestBank{
public static void main(String args[]){
Bank b=new SBI();//if object is PNB, method of PNB will be invoked
int interest=b.getRateOfInterest();
System.out.println("Rate of Interest is: "+interest+" %");
}}
```

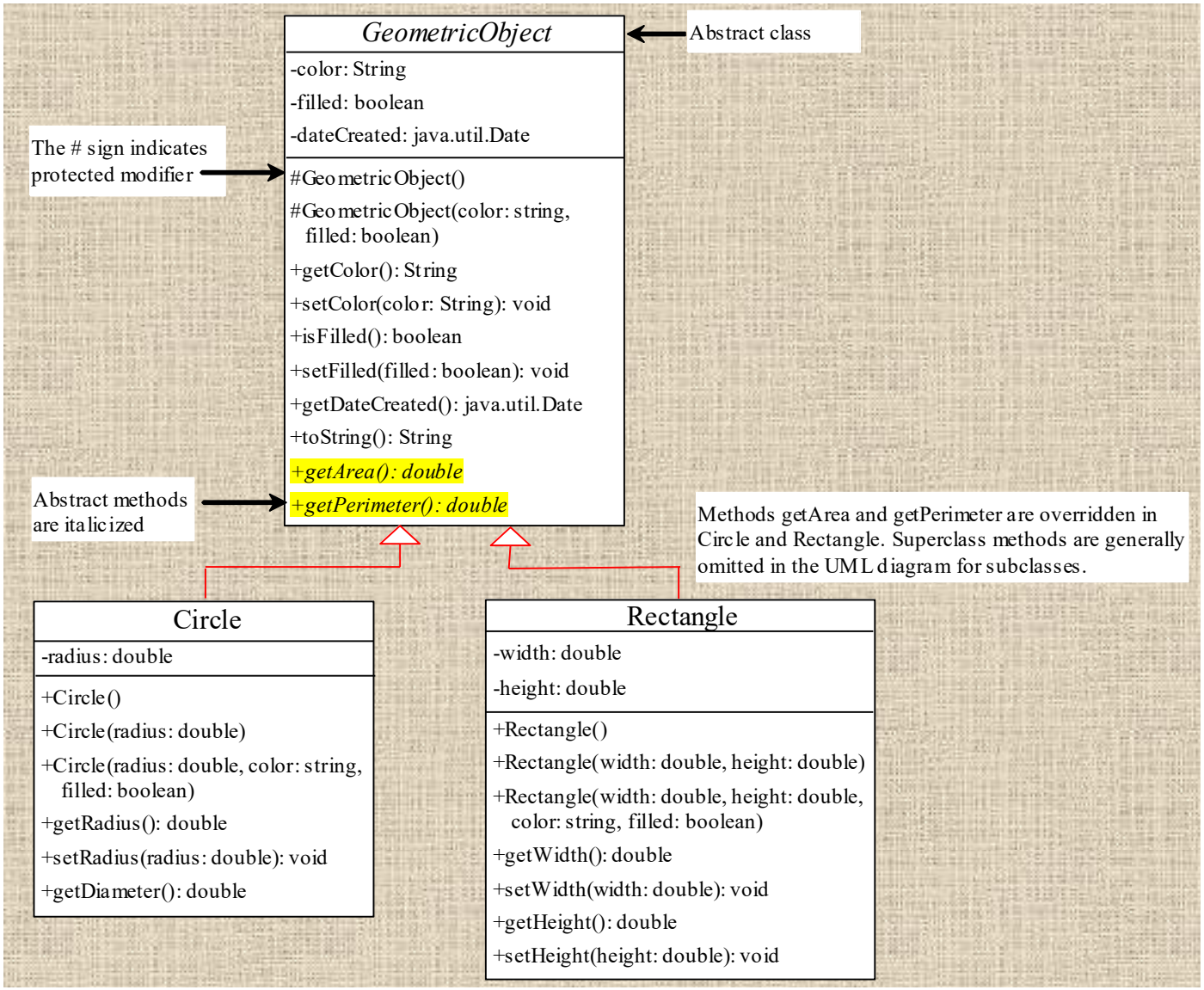
Test it Now

```
Rate of Interest is: 7 %
```

متد abstract در کلاس abstract

- از یک کلاس abstract نمی‌توان با استفاده از new شیئی ایجاد نمود.
- اما همچنان قادریم سازنده‌های آن را تعریف کرده و در سازنده‌های فرزندانشان فراخوانی کنیم.
- برای مثال، سازنده‌های GeometricObject در کلاس Circle و کلاس Rectangle فراخوانی می‌شوند.

متد abstract در کلاس abstract



نکته مهم در مورد کلاس abstract

- یک متد abstract نمی‌تواند درون یک کلاس غیر abstract قرار گیرد.
- اگر زیرکلاسی از یک ابرکلاس abstract تمامی متدهای abstract را پیاده‌سازی نکند، این زیرکلاس نیز باید abstract تعریف شود.
- به بیان دیگر، در یک زیرکلاس غیرabstract که از یک کلاس abstract ارث‌بری می‌کند، تمامی متدهای abstract باید پیاده سازی شوند، حتی اگر در زیرکلاس مورد استفاده قرار نگیرند.

کلاس abstract بدون متد abstract

- کلاسی با متدهای abstract باید abstract تعریف شود.
- اما می‌توانیم یک کلاس abstract بدون متدهای abstract تعریف کنیم.
- قادر نیستید با new نمونه (شیئی) از کلاس abstract ایجاد کنید.
- از این کلاس به عنوان کلاس والد برای تعریف زیرکلاسهای جدید می‌توان استفاده کرد.

ابرکلاس کلاس abstract می تواند یک کلاس عادی باشد؟

- یک زیرکلاس abstract می‌تواند والدی غیرabstract داشته باشد. برای مثال، کلاس Object کلاسی عادی (غیرabstract) است، اما هرکلاس abstract فرزند آن می‌باشد.

کلاس abstract به عنوان نوع

- با اینکه نمی‌توانیم با new از کلاس abstract شیئ بسازیم. اما یک کلاس abstract می‌تواند به عنوان یک نوع داده مورد استفاده قرار گیرد.

GeometricObject[] geo = new GeometricObject[10];

کلاس abstract با سازنده، فیلدهای داده‌ای و متدها

```
//example of abstract class that have method body
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }

 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

Test it Now

```
bike is created
running safely..
gear changed
```

نکته

- اگر در کلاسی متد abstract داشته باشیم، کلاس نیز باید حتماً abstract اعلان شود.

```
class Bike12{
abstract void run();
}
```

Test it Now

```
compile time error
```

- اگر بخواهید کلاس abstract را که حاوی متد abstract است، ارث‌بری کنیم باید پیاده‌سازی متد در کلاس فرزند ارایه شود یا کلاس فرزند نیز abstract تعیین شود.

یکی از کاربردهای کلاس abstract

- کلاس abstract می تواند برای پیاده‌سازی بخشهایی از interface مورد استفاده قرار گیرد. در این صورت، کاربر دیگر مجبور نیست همه متدهای interface را در کلاسهای معمولی پیاده‌سازی کند.

Interface در جاوا

- یک interface در جاوا یک نمایش (طرح کلی) از یک کلاس به نمایش می‌گذارد و تنها ثابتهای استاتیک و متدهای abstract در آن قرار می‌گیرند.
- Interface درجاوا سازوکاری است که با آن انتزاع به صورت کامل (100%) برقرار می‌شود. در interface تنها متدهای abstract که فاقد بدنه (پیاده‌سازی) هستند، قرار می‌گیرند.
- Interface به ما امکان می‌دهد انتزاع کامل داشته باشیم و بتوانیم ارث‌بری چندگانه را مدیریت کنیم.
- از Interface نیز مانند کلاس abstract امکان ایجاد نمونه (شیئ) وجود ندارد.

حذف سطوح دسترسی در interface

- در یک interface همه فیلدهای داده‌ای، استاتیک فاینال بوده و همه متدها، abstract می‌باشند.
- به این دلیل، این کلمات کلیدی را در هنگام تعریف interface می توان حذف نمود.

```
public interface T1 {
   public static final int K = 1;

   public abstract void p();
}
```

Equivalent

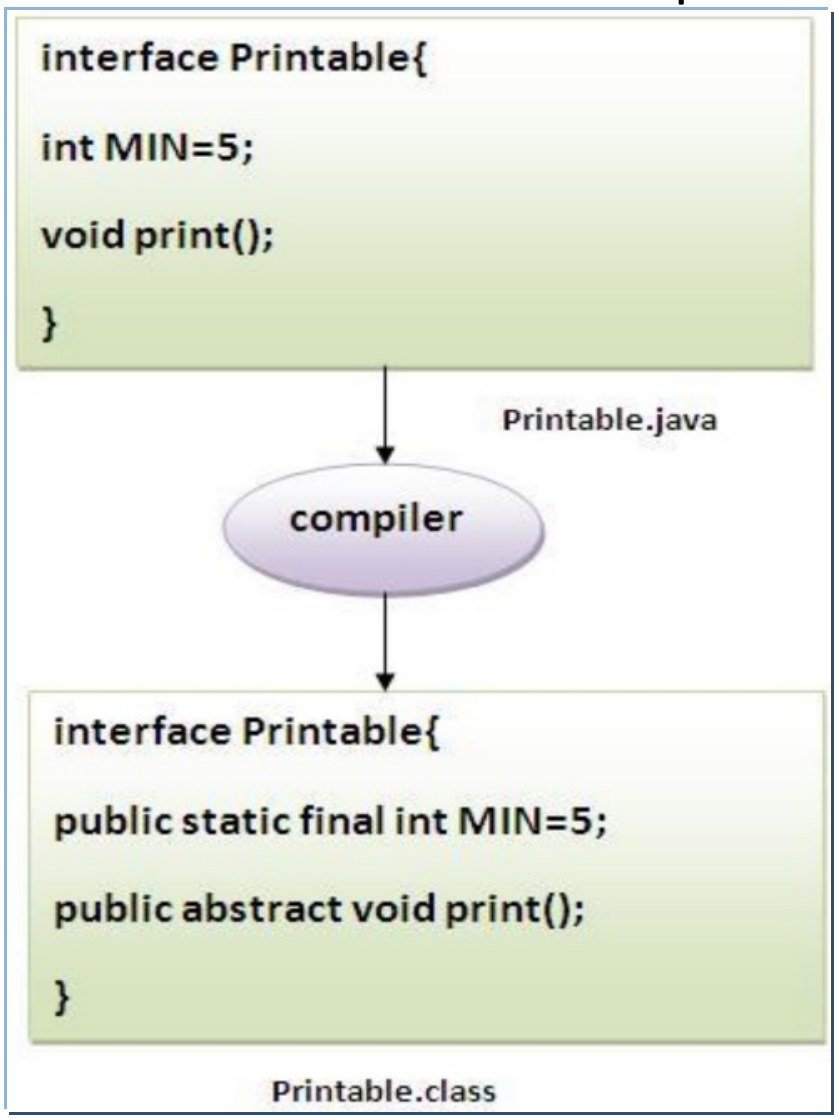
```
public interface T1 {
   int K = 1;

   void p();
}
```

- برای دسترسی به یک مقدار ثابت در interface به صورت زیر عمل می کنیم:

InterfaceName.CONSTANT_NAME (e.g., T1.K).

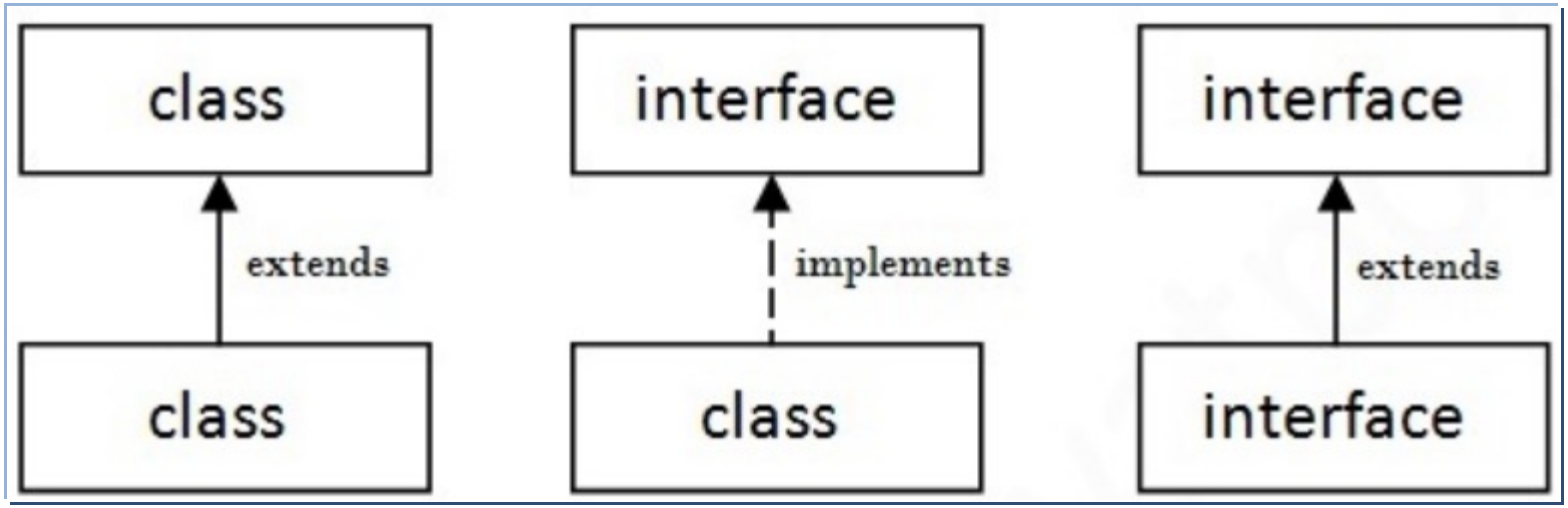
فیلدها و متدهای interface

- فیلدهای داده‌ای Interface به طور پیش‌فرض public، استاتیک و final هستند. متدها نیز public و abstract هستند.



فهم رابطه میان کلاسها و interface ها

- همانگونه که در شکل می‌بینید، یک کلاس از کلاسی دیگر ارث‌بری می‌کند.
- یک interface از یک interface دیگر ارث بری می‌کند.
- اما یک کلاس، یک interface را پیاده‌سازی می‌کند!



interface

- Interface در مواردی، مشابه کلاس abstract می‌باشد.
- اما تفاوتهایی عمده‌ای نیز دارد که مورد بررسی قرار خواهند گرفت.
- هدف از interface بیان رفتار اشیا است. برای مثال، با interface مشخص می‌کنیم که اشیا قابل خوردن، پرینت شدن، و غیره هستند.

تعریف یک Edible

برای اینکه بین interface و کلاس تمایز وجود داشته باشد، جاوا از کلمه کلیدی interface استفاده می کند:

```
public interface InterfaceName {
  constant declarations;
  method signatures;
}
```

مثال:

```
public interface Edible {
  /** Describe how to eat */
  public abstract String howToEat();
}
```

Interface یک کلاس خاص است

- با interface مانند یک کلاس ویژه در جاوا برخورد می شود.
- هر interface به یک فایل بایت کد مجزا (مانند یک کلاس معمولی) کامپایل می شود.
- مانند کلاس abstract نمی توان با استفاده از new از interface شیئی ایجاد نمود.
- اما در اکثر اوقات می توان از interface در کاربردهایی مشابه کاربردهای abstract استفاده نمود.
- برای مثال، شما می توانید از یک interface به عنوان یک نوع داده برای یک متغیر استفاده کنید.

مثال

- با یک interface به نام Edible (قابل خوردن) مشخص می کنیم که آیا شیئی خوردنی می باشد یا خیر؟
- به این منظور، کلاسی که می خواهیم بدانیم اشیای ایجاد شده از آن قابل خوردن هستند، با کلمه کلیدی implements این interface را پیاده سازی می کنند.
- برای مثال، کلاسهای Chicken و Fruit، خوردنی (edible) را که یک interface است، به صورتی که در اسلاید بعدی آمده پیاده سازی کرده اند:

مثال

```
public interface Edible {
  /** Describe how to eat */
  public abstract String howToEat();
}
```

```
public class TestEdible {
  public static void main(String[] args) {
    Object[] objects = {new Tiger(), new Chicken(), new Apple()};
    for (int i = 0; i < objects.length; i++)
      if (objects[i] instanceof Edible)
        System.out.println(((Edible)objects[i]).howToEat());    }
}
class Animal {
  // Data fields, constructors, and methods omitted here }
class Chicken extends Animal implements Edible {
  public String howToEat() {
    return "Chicken: Fry it";
  }
}
class Tiger extends Animal {
}
abstract class Fruit implements Edible {
  // Data fields, constructors, and methods omitted here
}
class Apple extends Fruit {
  public String howToEat() {
    return "Apple: Make apple juice";
  }
}
class Orange extends Fruit {
  public String howToEat() {
    return "Orange: Make orange juice";   }
}
```

مثالی ساده از interface در جاوا

- در این مثال یک interface به نام printable یک متد دارد و پیاده سازی آن توسط کلاس A انجام می گیرد.

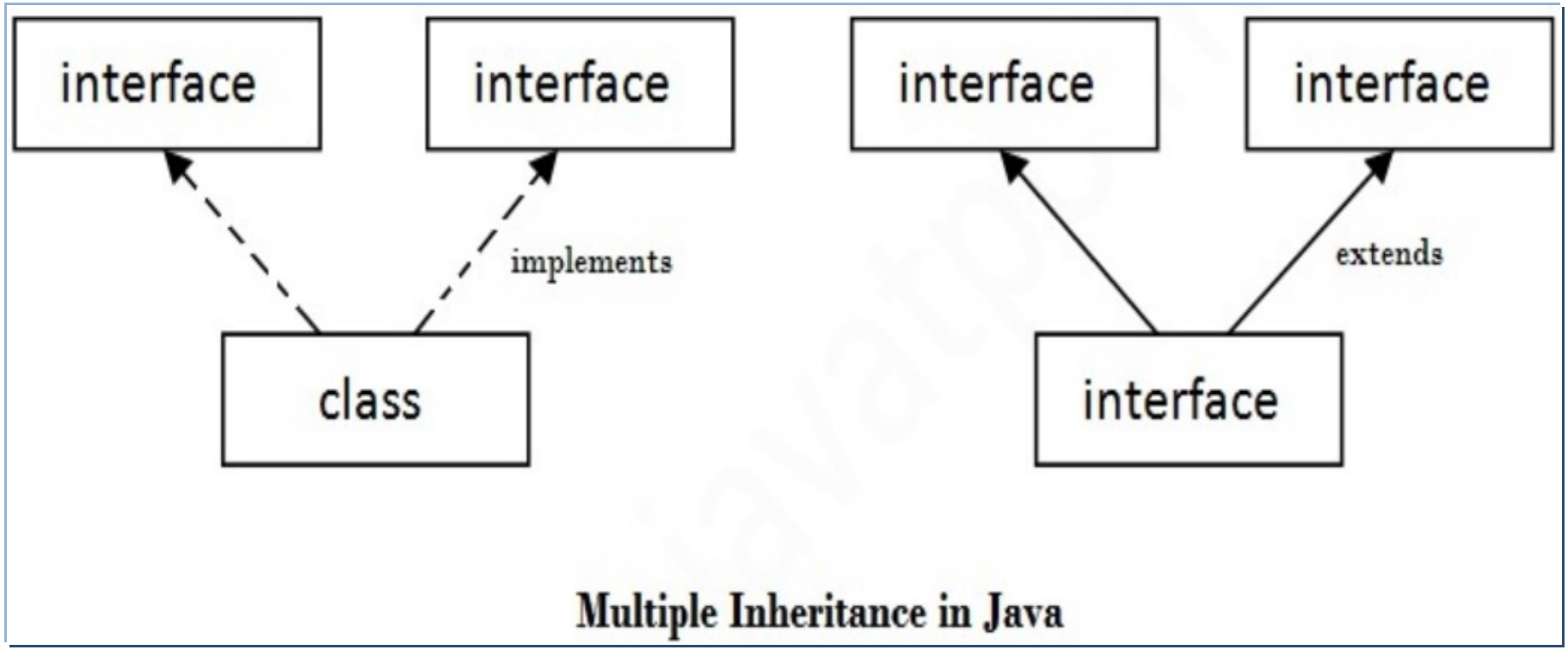
```
interface printable{
void print();
}

class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Test it Now

```
Output:Hello
```

ارث بری چندگانه در جاوا به کمک interface

- اگر کلاسی بیش از یک interface را پیاده سازی کند، یا یک interface از چندین interface ارث بری کند، می گوییم ارث بری چندگانه پشتیبانی شده است.



Multiple Inheritance in Java

ارث بری چندگانه در جاوا به کمک interface

```
interface Printable{
void print();
}

interface Showable{
void show();
}

class A7 implements Printable,Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Test it Now

```
Output:Hello
       Welcome
```

- در اسلاید ارث بری گفتیم، که ارث بری چندگانه در کلاسهای جاوا پشتیبانی نمی شود.
- اما در صورت استفاده از interface می توان ابهام را حذف نمود، چون پیاده سازی در interface ارایه نمی شود.
- مثال نشان داده شده در اسلاید بعد را مشاهده کنید:

مثال ارث بری چندگانه به کمک interface ها

```
interface Printable{
void print();
}

interface Showable{
void print();
}

class testinterface1 implements Printable,Showable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){
testinterface1 obj = new testinterface1();
obj.print();
 }
}
```

Test it Now

```
Hello
```

•همان طور که مشاهده می کنید، interfaceهای Printable و Showable دارای متدهای همنام هستند. اما پیاده سازی تنها در کلاس A وجود دارد. لذا ابهامی برای جاوا وجود نخواهد داشت!

ارث بری interface

- یک کلاس، یک interface را پیاده سازی می کند، اما یک interface می تواند از یک interface دیگر ارث بری داشته

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class Testinterface2 implements Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
Testinterface2 obj = new Testinterface2();
obj.print();
obj.show();
 }
}
```

Test it Now

```
Hello
Welcome
```

مثال: ترکیب abstract و interface

```
interface A{
void a();
void b();
void c();
void d();
}

abstract class B implements A{
public void c(){System.out.println("I am C");}
}

class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Test it Now

```
Output:I am a
       I am b
       I am c
       I am d
```

مقایسه abstract و interface

Interface	Abstract Class
• Methods can be declared	• Methods can be declared
• No method bodies	• Method bodies can be defined
• “Constants” can be declared	• All types of variables can be declared
• Has no constructors	• Can have constructors
• Multiple inheritance possible	• Multiple inheritance not possible
• Has no top interface	• Always inherits from **`Object`**
• Multiple “parent” interfaces	• Only one “parent” class

marker or tagged interface?

- به interface بدون هیچ عضوی، اصطلاحاً مارکر یا برچسب خورده گفته می شود.
- برای مثال:
  - Serializable –
  - Cloneable –
  - Remote –
- این نوع interface برخی اطلاعات موردنیاز را برای JVM فراهم می کند تا JVM بتواند عملیات مناسب را انجام دهد.

```
//How Serializable interface is written?
public interface Serializable{
}
```

برخی از واسطهای محبوب در جاوا

- **Iterator:** برای پیمایش برروی گردایه ای (collection) از اشیا بدون نیاز به دانستن نحوه ذخیره سازی اشیای ذخیره شده در آن، برای مثال در list،bag یا set. توضیحات کامل در بخش Collections ارایه خواهد شد.

- **Cloneable**: برای ایجاد یک کپی از شیئ موجود از طریق متد **clone()** که در کلاس Object قرار دارد.

- **Serializable:** بسته بندی (pack) کردن چندین شیئ به گونه ای که بتوان آنها را از طریق شبکه انتقال داد یا برروی دیسک ذخیره سازی نمود. در مقصد این بسته مجددا به اشیای مبدا تبدیل می شود. توضیحات بیشتر در درس I/O ارایه خواهد شد.

- **Comparable:** به هدف ایجاد یک ترتیب کلی (total order) برروی اشیا

واسط Iterator

- واسط Iterator در پکیج java.util قرار دارد و هدف آن دسترسی به عناصر (اشیای) ذخیره شده در collections می باشد.

```
package java.util;
public interface Iterator {
    // the full meaning is public abstract boolean hasNext()
    boolean hasNext();
    Object next();
    void remove(); // optional throws exception
}
```

SN	Methods with Description
1	**boolean hasNext( )** Returns true if there are more elements. Otherwise, returns false.
2	**Object next( )** Returns the next element. Throws NoSuchElementException if there is not a next element.
3	**void remove( )** Removes the current element. Throws IllegalStateException if an attempt is made to call remove( ) that is not preceded by a call to next( ).

واسط Iterator

- iterator شیئی است که واسط Iterator را پیاده سازی کرده است.
- پیش از آنکه بخواهید اشیای یک collection را پیمایش کنید ابتدا باید یک شیئ iterator ایجاد کنید.
- اینکار با فراخوانی متد iterator() در شیئی collection حاوی عناصر ذخیره شده صورت می گیرد.
- تصور کنید شیئی iterator یک پیماینده است که قادر است درون شیئی که خود از عناصر مختلف تشکیل شده حرکت کند و هربار به یک عنصر آن دسترسی پیدا کند.
- مراحل این کار:
  - یک پیمانده iterator را ایجاد کنید تا در ابتدای شیئ collection شما قرار بگبرد. به این منظور در شیئ collectionتان متد iterator() را فراخوانی کنید.
  - حال پیمانده باید در درون شیئ شما حرکت کند تا به عناصر دخیره شده در آن دست پیدا کند. حلقه ای بسازید و شرط تکرار آن را با متد hasNext() معین کنید. تا زمانی که عنصری در شیئ پیموده نشده باشد این متد مقدا true برمی گرداند.
  - با فراخوانی متد next() به هر عنصر ذخیره شده دست پیدا خواهید کرد. نوع برگشتی این متد Object است. بنابراین باید یک downcasting به نوع عنصر ذ آن دسترسی داشته باشید.

SN	Methods with Description
1	**boolean hasNext( )** Returns true if there are more elements. Otherwise, returns false.
2	**Object next( )** Returns the next element. Throws NoSuchElementException if there is not a next element.
3	**void remove( )** Removes the current element. Throws IllegalStateException if an attempt is made to call remove( ) that is not preceded by a call to next( ).

واسط Cloneable

- کلاس X که واسط cloneable را پیاده سازی کرده است به کاربر می گوید اشیای این کلاس می توانند همانندسازی شوند.
- واسط خود تهی است و متد یا فیلدی ندارد.

```
// Car example revisited
public class Car implements Cloneable{
    private String make;
    private String model;
    private double price;
   // default constructor
   public Car() {
        this("", "", 0.0);
    }
   // give reasonable values to instance variables
   public Car(String make, String model, double price){
        this.make  = make;
        this.model = model;
        this.price = price;
    }
    // the Cloneable interface
    public Object clone(){
        return new Car(this.make, this.model, this.price);
    }
}
```