کارگاه برنامه نویسی پیشرفته

دستور کار شماره شش

اهداف

آشنایی با ارث‌بری[1] در جاوا

آشنایی با Upcasting و Downcasting

آشنایی با کلاس‌های انتزاعی[2]

آشنایی با اینترفیس‌ها[3]

[1] Inheritance
[2] Abstract Classes
[3] Interfaces

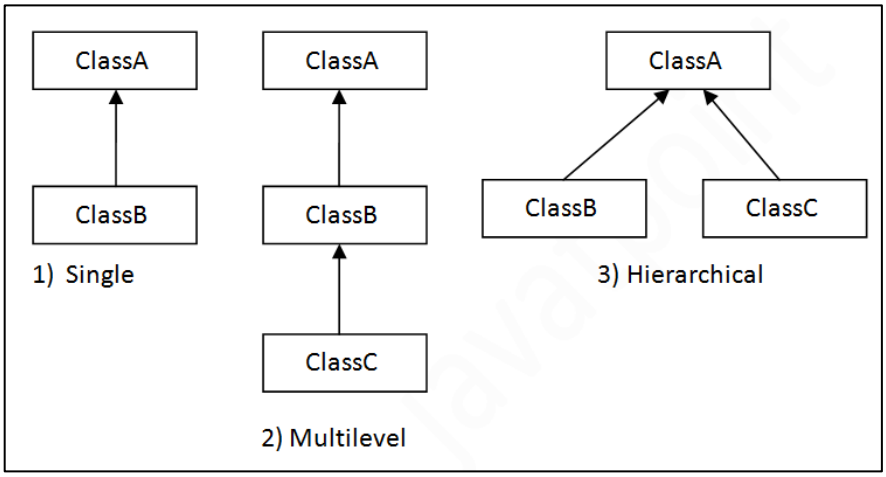
فهرست مطالب

ارث‌بری در جاوا

تعریف ارث‌بری

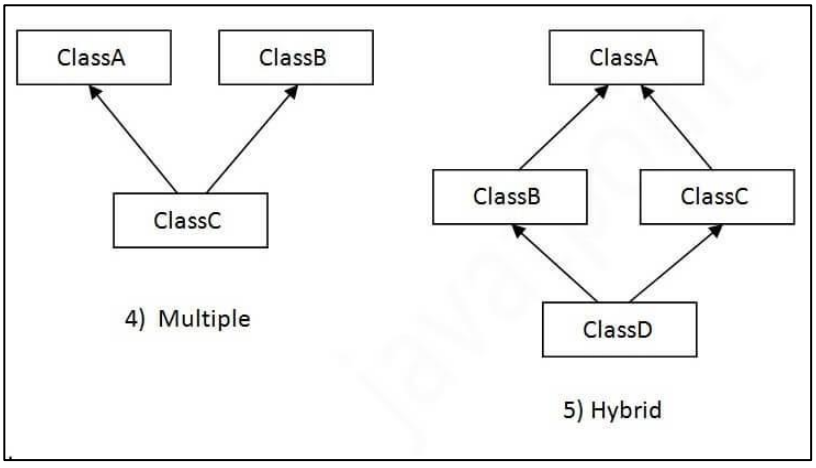
وراثت در جاوا مکانیزمی است که در آن یک شیء تمام خصوصیات و رفتارهای یک شیء والد را به دست می‌آورد. ایده‌ی کلی وراثت این است که می‌توان کلاس‌های جدیدی ایجاد کرد که بر اساس کلاس‌های موجود ساخته شده‌اند. وقتی از یک کلاس موجود ارث‌بری می‌کنیم، می‌توان از متدها و فیلدهای کلاس والد استفاده مجدد کرد. علاوه بر آن، می‌توان متدها و فیلدهای جدیدی را در کلاس فعلی نیز اضافه کرد.

از مهم‌ترین دلایل استفاده از ارث‌بری در جاوا می‌توان به چندریختی بودن در زمان اجرا و امکان استفاده‌ی مجدد کد اشاره کرد. ارث‌بری‌های زیر در جاوا پشتیبانی می‌شوند:



(انواع ارث‌بری‌های پشتیبانی شده در جاوا)

همچنین دو ارث‌بری زیر در جاوا پشتیبانی نمی‌شوند:



(انواع ارث‌بری‌های پشتیبانی نشده در جاوا)

سناریویی را در نظر بگیرید که A، B و C سه کلاس هستند. کلاس C کلاس A و B را به ارث می‌برد. اگر کلاس‌های A و B دارای یک متد باشند و آن را از شیء کلاس فرزند فراخوانی کنیم، ابهامی برای متد کلاس A یا B وجود خواهد داشت. از آنجا که خطاهای زمان کامپایل بهتر از خطاهای زمان اجرا هستند، هنگامی که کلاسی از دو کلاس دیگر ارث برده باشد (حتی در صورتی که کلاس‌ها، متدهای متمایزی داشته باشند)، جاوا در زمان کامپایل خطا بر می‌گرداند. در ادامه‌ی دستور کار، بیشتر با این مشکل آشنا خواهیم شد.

کلیدواژه‌ی super

در جاوا کلیدواژه‌ی super برای کلاس‌هایی که از ارث‌بری استفاده می‌کنند، کاربردهای زیادی دارد که در اینجا مهم‌ترین آن‌ها یعنی دسترسی به کانستراکتور کلاس والد را بررسی می‌کنیم.
برای مثال فرض کنید یک کلاس Parent به شکل زیر داریم:

```
class Parent {
    private int anInt;
    public Parent(int anInt){
        this.anInt = anInt;
    }

    public int getAnInt() {
        return anInt;
    }
}
```

(کلاس Person)

همچنین یک کلاس Child داریم که از این کلاس ارث‌بری می‌کند:

```
class Child extends Parent{
    private double aDouble;

    public Child(int anInt,double aDouble){
        this.aDouble = aDouble;
    }

    public double getaDouble() {
        return aDouble;
    }
}
```

(کلاس Child)

با توجه به پرایوت بودن فیلد anInt کلاس والد، نمی‌توان آن را در کانستراکتور کلاس Child مقداردهی کرد. حال راه‌حل این مشکل استفاده از کلیدواژه‌ی super در کانستراکتور کلاس Child برای دسترسی به کانستراکتور کلاس والد خود است.

کد کامل و درست این کلاس:

```
class Child extends Parent{
    private double aDouble;

    public Child(int anInt,double aDouble){
        super(anInt);
        this.aDouble = aDouble;
    }

    public double getaDouble() {
        return aDouble;
    }
}
```

(نحوه‌ی صحیح پیاده‌سازی کلاس Child)

چند نکته

- در هر صورت نوشتن دستور super() در کانستراکتور کلاس‌های فرزند وجود دارد. البته در صورتی که در کلاس والد، کانستراکتوری بدون آرگومان داشته باشیم، خود کامپایلر جاوا آن را درج می‌کند.
- این دستور باید اولین دستور در کانستراکتور کلاس فرزند باشد.

کلیدواژه‌ی instanceof

کلیدواژه‌ی instanceof، دستوری است که بررسی می‌کند شیء داده شده، از نوع کلاس مورد نظر هست یا خیر. برای مثال، همان کلاس‌های Parent و Child در مثال بالا را در نظر می‌گیریم:

```
public class Main {
  public static void main (String[] args) {
    Child child = new Child (10, 0.1);

    if (child instanceof Child)
        System.out.println("child is instanceof Child Class");

    if (child instanceof Parent)
        System.out.println("child is instanceof Child Class");

    Parent parent1 = new Parent (10);
    if (!(parent1 instanceof Child))
        System.out.println("parent1 is not instanceof Child Class");

    Parent parent2 = null;
    if (!(parent2 instanceof Parent))
        System.out.println("instanceof always return false for null");
  }
}
```

(مثال استفاده از کلیدواژه‌ی instanceof)

چند نکته

- همواره یک شیء ساخته شده، از نوع کلاس خود است.
- یک شیء ساخته شده، از نوع کلاس والد خود نیز است.
- یک شیء ساخته شده، از نوع کلاس فرزند خود نیست.
- null از نوع هیچ کلاسی نیست.

آشنایی با Upcasting و Downcasting

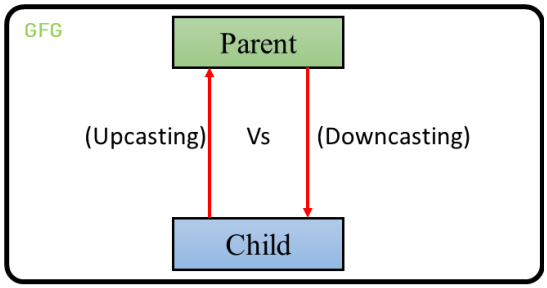
معرفی دو اصطلاح

با مفهوم کست کردن[1] در گذشته آشنا شده‌اید. حال می‌خواهیم با دو مفهوم Upcasting و Downcasting آشنا شویم.

برای درک بهتر، همان مثال کلاس‌های Child و Parent در مثال‌های بالا را در نظر می‌گیریم. حال دو اصطلاح را تعریف می‌کنیم:

Upcasting: اگر بخواهیم یک شیء از کلاس فرزند (در اینجا Child) را به کلاس پدر (در اینجا Parent) کست کنیم، به این کار Upcasting گفته می‌شود.

Downcasting: اگر بخواهیم یک شیء از کلاس پدر (دراینجا Parent) را به کلاس فرزند (در این‌جا Child) کست کنیم، به این‌کار Downcasting گفته می‌شود.



(مفاهیم Downcasting و Upcasting)

[1] Casting

```
Parent p = new Child();
```
(مثال Upcasting)

```
Child c = (Child)p;
```
(مثال Downcasting)

توجه کنید همان‌طور که مشاهده کردید، در upcasting به دلیل اینکه هر شیء از جنس فرزند قطعاً از نوع جنس پدر است، کست به صورت ضمنی[1] انجام می‌شود و نیازی به کست کردن صریح[2] نیست. اما در downcasting باید به این نکته توجه کرد که شیء سمت راست حتماً از جنس کلاس سمت چپ باشد در غیر این صورت با خطای ClassCastException مواجه می‌شویم.

برای مثال فرض کنید کست مقابل را انجام دهیم:

```
Object o1 = new Object();
String s1 = (String)o1;
```

از آن‌جایی که کلاس String از کلاس Object ارث‌بری می‌کند و در این مثال، شیء o1 از جنس Object و قابل کست کردن به String نیست، با خطا مواجه می‌شویم.

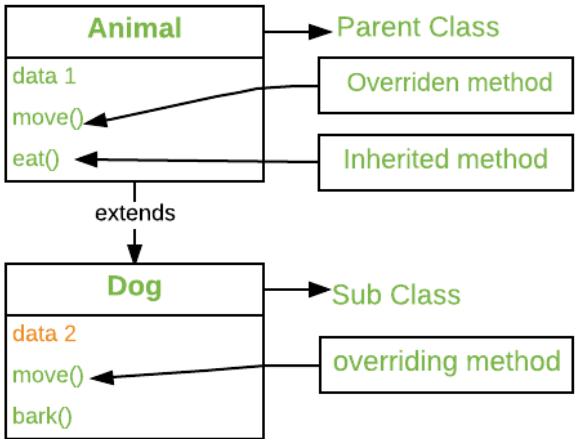
[1] Implicit
[2] Explicit

آشنایی با Method Overriding

معرفی Method Overriding

در زبان‌های برنامه‌نویسی شی‌گرا و همچنین جاوا، overriding یک ویژگی در ارث‌بری است که به فرزندها اجازه می‌دهد یک پیاده‌سازی خاص برای یک متد از پیش تعریف شده در کلاس پدر، ارائه دهند.

به عبارتی دیگر، اگر متدی در کلاس فرزند پیاده‌سازی شود که نام، آرگومان‌ها، سطح دسترسی و ریترن‌تایپ[1] یکسانی با متدی در کلاس پدر داشته باشد، گفته می‌شود که آن متد override شده است:



(توضیح method overriding)

حال این سؤال پیش می‌آید که در هنگام صدا زدن، کدام یک از متدها صدا زده می‌شود؟ متدی که در کلاس فرزند وجود دارد یا متدی که در کلاس پدر قرار دارد؟ به مثال زیر توجه کنید:

[1] Return Type

```java
class Parent {
    void show()
    {
        System.out.println("Parent's show()");
    }
}
class Child extends Parent {
    @Override
    void show()
    {
        System.out.println("Child's show()");
    }
}
```

(پیاده‌سازی دو کلاس Child و Parent)

همان‌طور که مشاهد می‌کنید، کلاس Child از Parent ارث‌بری کرده و متد show را override کرده است.

```java
public class Main {
    public static void main(String[] args) {
        Parent obj1 = new Parent();
        obj1.show();

        Parent obj2 = new Child();
        obj2.show();
    }
}
```

(پیاده‌سازی کلاس Main)

خروجی کد بالا به صورت زیر خواهد بود:

- Parent’s show()
- Child’s show()

دقت کنید از آن‌جایی که obj1 از جنس Parent بود، متد show در کلاس Parent اجرا شد و به دلیل مشابه، در obj2 متد show در کلاس Child اجرا شد.

در حالت کلی در جاوا، هنگام صدا زدن هر متد، ابتدا بررسی می‌شود که متد در کلاس فرزند وجود داشته باشد. اگر وجود داشت، اجرا می‌شود و در غیر این صورت به سراغ کلاس پدر رفته و همین فرآیند را برای کلاس پدر نیز تکرار می‌کند.

کلاس‌های انتزاعی[1]

معرفی کلاس‌های انتزاعی

کلاس انتزاعی، کلاسی‌ست که بخشی از بدنه‌ی آن تعریف نشده است و کلاس‌هایی که از آن ارث می‌برند ملزم به تکمیل متدهای انتزاعی آن هستند.

برای مثال، برای مدل‌سازی یک بازی شطرنج، می‌توان یک کلاس Piece برای مهره‌ها به شکل انتزاعی تعریف نمود که شامل متد move به شکل انتزاعی باشد. واضح است که نحوه‌ی حرکت مهره‌های گوناگون، متفاوت است و برای هر یک از کلاس‌های مربوط به مهره‌ها که از این کلاس ارث می‌برند، می‌توان متد move را به شکل متفاوتی پیاده‌سازی کرد.

برای تعریف کلاس Piece به شکل انتزاعی، در هنگام تعریف کلاس باید از کلیدواژه‌ی abstract استفاده کنیم:

```
public abstract class Piece {...}
```

همچنین در ابتدای تعریف متدهای مورد نظر که باید توسط کلاس‌های فرزند override شوند نیز، باید از این کلیدواژه استفاده کرد:

```
public abstract void move();
```

برای متدهای انتزاعی، بدنه‌ای تعریف نمی‌شود. پس از تعریف ورودی‌های متد در پرانتز، دستور با علامت سمیکالن به پایان می‌رسد و نمی‌توان برای آن بدنه‌ای تعریف کرد.

ارث‌بری از کلاس انتزاعی نیز تفاوتی با دیگر کلاس‌ها ندارد و با استفاده از کلیدواژه‌ی extends انجام می‌شود:

```
class Queen extends Piece {...}
```

همچنین دقت کنید که نمی‌توان شیءای از یک کلاس انتزاعی ساخت. به این دلیل که کلاس‌های انتزاعی اشاره به مفاهیمی دارند که در کلاس انتزاعی تعریف درستی از آن‌ها قابل ارائه نیست و اما فرزندان این کلاس امکان ارائه تعریف را دارند. برای مثال در برنامه‌ی شطرنج، هیچ یک از اشیاء از نوع Piece نخواهند بود و از کلاس‌هایی که از Piece ارث‌بری می‌کنند، استفاده می‌کنیم.

[1] Abstract Classes

کاربرد کلاس‌های انتزاعی، استفاده‌ی آن‌ها در چندریختی‌ها است. برای مثال در مثال شطرنج، می‌توان متد move مربوط به مهره‌ای که کاربر انتخاب کرده و به عنوان متغیری از نوع Piece ذخیره شده را فراخوانی کرد، بی آنکه نیاز به بررسی نوع مهره‌ی مد نظر باشد.

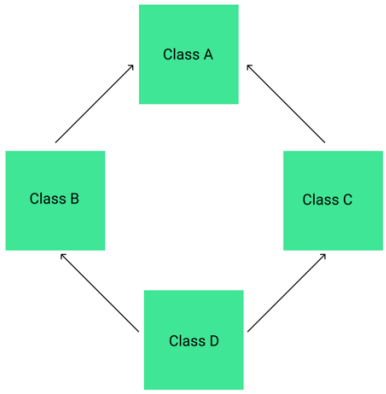
اینترفیس‌ها[1]

معرفی اینترفیس‌ها

گاهی اوقات، بسته به طراحی مورد نیاز در برنامه، ممکن است نیاز به کلاس‌هایی داشته باشیم که ویژگی‌های مشترکی را با بیش از یک دسته از دیگر کلاس‌ها داشته باشند؛ برای این‌کار شاید در نگاه نخست این‌طور به نظر برسد که کلاس مورد نظر می‌تواند بیش از یک والد داشته باشد، اما به دلیل وجود چالشی معروف به « diamond problem»، یک کلاس نباید بیش از یک کلاس والد داشته باشد. اما چالش diamond چیست؟

معرفی Diamond Problem

همان‌طور که می‌دانید، در هنگامی که متدی از یک شیء را فراخوانی می‌کنیم، مفسر جاوا ابتدا به بررسی وجود متد در کلاس سازنده‌ی آن شیء می‌پردازد و در صورتی که متد مورد نظر در آن موجود نباشد، به سراغ کلاس والد رفته همین روند را برای کلاس والد انجام می‌دهد تا به کلاسی برسد که شامل تعریف متد مورد نظر باشد. حال در صورت وجود بیش از یک والد، ممکن است متد مورد نظر در هر دو والد تعریف شده باشد و در جاوا نمی‌توان اولویتی برای آن در نظر گرفت که متد فراخوانی شده بر اساس کدام تعریف اجرا شود، بنابراین نمی‌توان بیش از یک والد برای یک کلاس در جاوا تعریف کرد.



[1] Interfaces

اینترفیس‌ها در جاوا

اما راه‌حل جاوا برای طراحی کلاس‌هایی که ویژگی‌های مشترکی با بیش از یک دسته از کلاس‌ها دارند چیست؟ پاسخ اینترفیس است. اینترفیس مفهومی مشابه کلاس است، با این تفاوت که نمی‌توان برای متد های آن بدنه‌ای تعریف کرد و به نوعی، یک قرارداد برای تعریف کلاس‌هایی با ویژگی‌های مشترک است.

به عنوان مثال در برنامه شطرنج، علاوه بر کلاس Piece که همه کلاس‌های مربوط به مهره‌ها از آن ارث‌بری می‌کنند، می‌توان اینترفیس‌های مربوط به مهره‌های سفید و سیاه را نیز پیاده‌سازی کرد تا هر یک از انواع مهره‌ها بسته به این ویژگی خود، یکی از آن‌ها را ارث‌بری کنند.

برای تعریف مفهوم گفته شده، از کلیدواژه interface استفاده می‌شود:

```
public interface WhitePiece {...}
```

در هنگام تعریف متدهای اینترفیس، مشابه متدهای انتزاعی نمی‌توان بدنه‌ای پیاده‌سازی کرد، اما نیاز به استفاده از کلیدواژه‌ی خاصی نیست.

همچنین برای پیاده‌سازی اینترفیس‌ها، در تعریف کلاس مورد نظر باید از کلیدواژه‌ی implements استفاده کرد:

```
class Queen extends Piece implements WhitePiece {...}
```

برخلاف کلیدواژه‌ی extends که پس از آن تنها یک کلاس را می‌توان نام برد، هنگام پیاده‌سازی اینترفیس‌ها می‌توان بیش از یک اینترفیس را پس از implements ذکر کرد که برای جداسازی آنها از علامت «,» استفاده می‌شود:

```
class A implements B, C {...}
```

انجام دهید

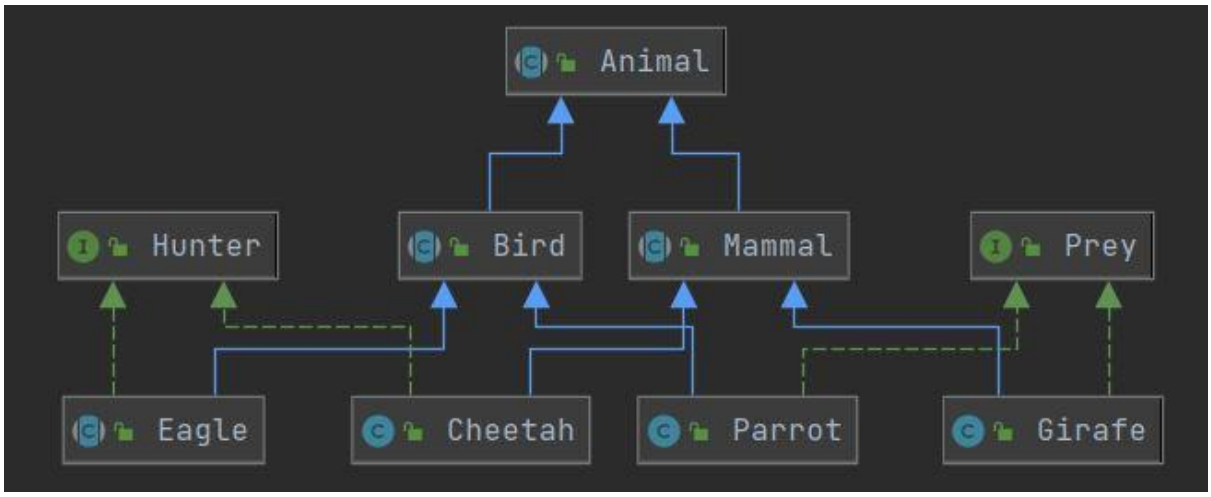
شبیه‌سازی مدیریت باغ‌وحش

در این تمرین می‌خواهیم یک سیستم مدیریت باغ وحش را شبیه‌سازی کنیم. این سیستم، ورود و خروج حیوانات و وضعیت حیوانات باغ‌وحش را کنترل می‌کند. برای پیاده‌سازی این سیستم نیاز به استفاده از مفاهیم ارث‌بری و چندریختی داریم (در این تمرین، بعد از انجام هر بخش، پول ریکوئست بخش مربوطه را ساخته و سپس بخش بعدی را انجام دهید).

ابتدا یک ریپازیتوری با نام «AP-Workshop6» بسازید.

بخش اول

برای مثال طراحی زیر را در نظر بگیرید:



(طراحی باغ‌وحش)

در این دیاگرام، کلاس‌های Animal، Bird و Mammal کلاس‌های انتزاعی هستند.

pull request 1: (اپلود کردن طراحی کلاس‌ها و سلسله مراتب ارث‌بری‌ها، بدون پیاده‌سازی کلاس‌ها)

بخش دوم

هرکدام از این حیوانات ویژگی‌های مشترکی با دیگر حیوانات و ویژگی‌های منحصر به فردی نیز دارند. هدف نهایی این تمرین این است که از هر کدام از این حیوانات یک یا چند شیء ایجاد کنید و آن‌ها را در لیستی ذخیره کنید و در انتها متد show را در هریک را صدا بزنید.

در پیاده‌سازی دقت کنید که هر حیوان یک نام و یک سن دارد. علاوه بر آن، هر پرنده یک ارتفاع پرواز دارد و هر پستاندار یک سرعت دویدن دارد. همچنین هر حیوان یک فیلد از نوع string دارد که به بیان ویژگی‌های منحصربه‌فرد آن حیوان می‌پردازد و باید برای هر حیوان متمایز باشد. تمام ویژگی‌های حیوانات در هنگام ساخت در کانستراکتور باید مقداردهی شوند.

متد show باید عبارتی با قالب زیر را در کنسول چاپ کند:

Row. name, age, height of fly/speed, animal special

برای مثال:

3. name: Cheetah, age: 12, speed: 120, “The fastest”

pull request 2: (پیاده سازی و قراردادن فیلد ها و متد های مورد نیاز هر کلاس)

بخش سوم

همچنین اینترفیس‌های Hunter و Prey نیز باید پیاده‌سازی شوند که اشاره به حیوانات شکارچی و یا شکارشونده دارند و هر یک از حیوانات با توجه به ماهیتی که دارند، یکی از این اینترفیس‌ها را نیز پیاده‌سازی می‌کنند.

اینترفیس Hunter باید شامل متد hunt باشد که به عنوان ورودی یک Prey می‌گیرد و در کنسول، عبارتی شامل نام شکارچی و نام شکار چاپ می‌کند.

برای مثال:

Cheetah hunted Parrot

برای این منظور، باید اینترفیس Prey شامل متد getName باشد تا در متد hunt، بتوان نام حیوان موجود در آرگومان متد را در خروجی چاپ کرد.

pull request 3: (طراحی منو یا قراردادن تست کیس ها داخل کد)