# آموزش مقدماتی برنامه نویسی به زبان جاوا

مدرس : سمیه اوجی

# آرایه ها

آرایه مجموعه ای از داده های هم نوع است که در حافظه در آدرس های پشت سرهم و تحت یک نام مشترک ذخیره و مقدار دهی می شوند.

تعریف آرایه یک بعدی  نوع آرایه[] نام آرایه= new نوع آرایه[طول];

Exp)

تعریف آرایه ای به طول ۱۰ از نوع اعداد صحیح ←
```
int[] ArrInt=new int[10];
```
تعریف و مقدار دهی آرایه ای به طول ۵ از نوع اعداد صحیح←
```
int[] Arr1={12,3,5,10,13};
```

تعریف و مقدار دهی آرایه ای از نوع رشته ای ←
```
String[] name = new String[3];
name[0] = "Iman";
name[1] = "Aria";
name[2] = "Mohammad Ali";
```

تعریف آرایه دو بعدی

نوع آرایه[][] نام آرایه= new نوع آرایه[تعداد سطر][تعداد ستون];

Exp )

```
int[][] twodimArr=int [3][8] ;
```

```
int[][] row = {{10, 20, 30}, {100, 200, 300}};
row[1][2]=400;
```

عملیات روی آرایه

برای مرتب سازی آرایه Arrays.sort(نام آرایه)

به دست آوردن طول آرایه نام آرایه.length

## متد (method)

متد شامل مجموعه ای از  کدهاست که وظیفه خاصی را دنبال می کند.

- متد در داخل کلاس تعریف می شود.
- نمی توان یک متد را در  داخل متد دیگر تعریف کرد.

( ... ، نام پارامتر۲ نوع پارامتر ورودی۲،نام پارامتر۱ نوع پارامتر ورودی۱)نام متد  نوع مقدار بازگشتی متد

```
{
    کدهای اجرایی;
    return مقدار بازگشتی;
}
```

Exp)

```
double clacSum (double a , double b)
{
    return (a+b);
}
```

## متد بازگشتی (Recursion method)

متد بازگشتی متدی است که به طور متناوب خود را صدا می زند.

Exp)

```
long fact(int n)
{
    if (n==1)
        return 1;
    else
        return n*fact(n-1);
}
```

متدی بنویسید که عدد صحیحی را از ورودی گرفته و تعداد ارقام آن را برگرداند.
(راهنمایی : تعداد ارقام یک عدد را می توان با تقسیم های متوالی بر ۱۰ به دست آورد. )

متد را یکبار به روش عادی و یک بار به روش بازگشتی بنویسید.

## روش ارسال پارامتر به متد در جاوا

- ✓ برای متغیر های معمولی (غیر آرایه ای) ارسال پارامتر به روش مقدار است. یعنی مقدار پارامتر ارسالی در آرگومان کپی می شود و با تغییر دادن آن در درون متد، تاثیری در متغیری که آن را فراخوانی کرده ندارد.

- ✓ برای متغیرهای آرایه ای، مقدار دهی به روش ارجاع است و تغییر در مقادیر آرایه در درون متد، پس از خروج از متد نیز همچنان پا برجاست.

## سربارگذاری متد (method overloading)

سربارگذاری متد به ما این امکان را می دهد که چندین متد با نام یکسان اما پارامترهای ورودی متفاوت، کدهای اجرایی متفاوت و نوع خروجی متفاوت داشته باشیم. در زمان اجرا، برنامه به طور خودکار از آرگومان های ارسال شده به آن تشخیص می دهد که کدام متد مد نظر شماست.

Exp)

```
int mysum(int a, int b)
{
    return (a+b);
}

double mysum(double a, double b)
{
    return (a+b);
}

int mysum(int[] arr)
{
    int s=0;
    for(int a:arr)
        s+=a;
    return s;
}
```

# برنامه نویسی شی گرا (object oriented programming)

برنامه نویسی شی گرا روشی در برنامه نویسی است که به خوانا تر بودن و راحت تر نوشتن برنامه کمک می کند. قابلیت استفاده مجدد یکی از قابلیت های مهم این نوع برنامه نویسی است.

- ✓ جاوا یک زبان برنامه نویسی شی گرا است.
- ✓ در برنامه نویسی شی گرا همه چیز در قالب کلاس نوشته می شود.
- ✓ ابتدا کلاس نوشته شده و سپس شی از روی کلاس نمونه گیری می شود.
- ✓ برای مثال نقشه ساختمان یک کلاس و ساختمان ساخته شده از روی آن نقشه یک شی است.

## کلاس (class)

- ✓ کلاس مانند یک نوع داده است که توسط کاربر تعریف می شود.
- ✓ هر کلاس شامل فیلدها (یا خاصیت ها) و متدها است.
- ✓ شی نمونه واقعی ساخته شده از روی کلاس است.

فرم کلی تعریف کلاس در جاوا ←

```
نام کلاس  class سطح دسترسی
{
    تعریف فیلد ۱;
    تعریف فیلد ۲;
    .
    .
    .
    تعریف متد ۱;
    تعریف متد ۲;
    .
    .
    .
}
```

Exp)

```
public class myclass {
    public String name;
    public String family;
    public int age;
    public String AllInfomation()
    {
        String s;
        s="Name="+ name + " Family="+ family +" Age="+ age;
        return s;
    }
}
```

در جایی که می خواهیم از کلاسمان استفاده کنیم، کدهایی مانند کد های زیر خواهیم داشت:

```
myclass obj1=new myclass();
obj1.age=10;
obj1.name=“sara”;
obj1.family=“alavy”;
String str=  obj1.AllInfomation();
```

## متد سازنده کلاس (constructor)

سازنده کلاس متد خاصی است که به شما اجازه می دهد مقدار دهی اولیه خاصیت ها یا هر دستوری که نیاز است در زمان ایجاد شی اجرا شود را در آن بنویسیم. متد سازنده دارای دو ویژگی اصلی است:

۱) نام آن باید دقیقا همنام کلاس باشد.
۲) مقدار برگشتی ندارد و هیچ نوعی برای مقدار برگشتی آن نمی نویسم.

- ✓ اگر هیچ سازنده ای را برای کلاس تعریف نکنیم، کامپایلر یک سازنده پیش فرض، که یک متد بدون پارامتر و بدون کد است را به عنوان سازنده در نظر می گیرد.

**Exp)**

```
public myclass(String FirstName,String LastName,int YourAge)
{
    name=FirstName;
    family=LastName;
    age=YourAge;
}
```

- ✓ کلمه کلیدی this ← اگر در زمان نوشتن یک متد کلاس، پارامتر یا متغیر محلی ای هم نام با فیلدی از کلاس وجود داشت و هدفمان فیلد کلاس است از این کلمه کلیدی استفاده می کنیم.

**Exp)**

```
public class myclass {
    public String name;
    public String family;
    public int age;
    public String AllInfomation()
    {
        String s;
        s="Name="+ name + " Family="+ family +" Age="+ age;
        return s;
    }
    public myclass(String FirstName,String LastName,int age)
    {
        name=FirstName;
        family=LastName;
        this.age=age;
    }
}
```

# سطوح دسترسی

private ← اگر یک عضو از یک کلاس به صورت private تعریف شود؛ آن عضو فقط در کد نویسی(تعریف) کلاس قابل دسترسی است.

public ← اگر یک عضو از یک کلاس به صورت public تعریف شود، آن عضو در همه جا قابل دسترسی است. (در همان کلاس، سایر کلاس ها، زیر کلاس های ارث بری کننده از کلاس و ...)

protected ← گر یک عضو از یک کلاس به صورت protected تعریف شود، در همان کلاس و کلاس هایی که از آن کلاس ارث بری می کنند قابل دسترسی است.

سطح دسترسی پیش فرض ← اگر هیچ سطح دسترسی ای برای یک عضو از یک کلاس تعریف نکنیم، آن عضو دارای یک سطح دسترسی پیش فرض می شود که شبیه حالت protected است.

| | در داخل کلاس | سایر کلاس های همان package | زیر کلاس های همان package | زیر کلاس ها در سایر packageها | سایر کلاس ها در سایر packageها |
|---|---|---|---|---|---|
| public | √ | √ | √ | √ | √ |
| private | √ | X | X | X | X |
| protected | √ | √ | √ | √ | X |
| پیش فرض | √ | √ | √ | X | X |

- ✓ Package ← برای دسته بندی یا گروه بندی کلاس ها استفاده می شود.
- ✓ منظور از زیر کلاس، کلاسی است که از یک کلاس والد ارث بری کرده است.

```
public class accountClass {
    public String FirstName;
    public String LastName;
    public String AccountId;
    private long Amount;
    public accountClass(String fName,String lName,String
    accId)
    {
        FirstName=fName;
        LastName=lName;
        AccountId=accId;
        Amount=0;
    }
    public void Variz(long mablagh)
    {
        Amount+=mablagh;
    }
    public void Bardasht(long mablagh)
    {
        if (Amount>=mablagh)
        Amount-= mablagh;
        else
        System.out.println("mojoudi kafi nist.");
    }
    public long Mojoudi()
    {
        return Amount;
    }
}
```

Exp) یک کلاس تعریف کنید که اطلاعات حساب یک فرد را نگه دارد و عملیات مورد نظر را انجام دهد.

اطلاعات شامل:

- نام
- نام خانوادگی
- شماره حساب
- موجودی حساب

و

عملیات ها :

- واریز وجه
- برداشت وجه
- دریافت موجودی

کلاسی برای کار بر روی یک داده صحیح بنویسید بطوری که متدهایی برای انجام عملیات های زیر را داشته باشد.

- ✓ تعداد ارقام عدد را به دست آورد.
- ✓ نشان دهد که عدد داده شده مثبت است یا نه.
- ✓ نشان دهد که عدد اول است یا نه.

## ارث بری (inheritance)

اگر برنامه نویس بخواهد کلاسی بسازد که تمام یا برخی از اعضای یک کلاس موجود را دارا باشد می تواند از قابلیت ارث بری استفاده کند.

- ✓ extends ← برای ارث بری کلاس فرزند از کلاس پدر از کلمه کلیدی extends استفاده می کنیم.
- ✓ @override ← برای بازنویسی متدی از کلاس پدر در کلاس فرزند، از عبارت @override به همراه نام و ساختار کلی متد مورد نظر استفاده می شود. نکته ای که باید در نظر گرفت این است که ما فقط می توانیم کدهای یک متد را بازنویسی کنیم و امکان تغییر تعداد یا نوع پارامترها وجود ندارد.

Exp) حال فرض کنید که می خواهیم علاوه بر کلاس حساب قبل، نوع خاصی از حساب داشته باشیم که امکان برداشت بیش از ۵۰۰۰۰ تومان از حساب خود را نداشته باشد.

```
public class LimitedAccountClass extends accountClass{
    public LimitedAccountClass(String fName,String lName,String accId)
    {
        super(fName,lName,accId);
    }
    @Override
    public void Bardasht(long mablagh) {
        if (mablagh>50000)
            System.out.println("saghf bardasht roayat nashode.(shghfe bardasht 50000 mibasha.)");
        else
            super.Bardasht(mablagh);
    }
}
```

کلمه کلیدی super ← اگر در کلاس فرزند عنصری همنام با یک عنصر در کلاس پدر داریم و منظور ما عنصر کلاس پدر است، برای دسترسی به آن عنصر از کلمه کلیدی super استفاده می شود. همچنین برای فراخوانی سازنده کلاس پدر از کلمه کلیدی، (پارامترها)super استفاده می شود.

## مدیریت استثنا ها(Exceptions) و خطایابی

جاوا دارای مجموعه ای بزرگ از کلاس های مدیریت استثنا است، که برنامه نویس می تواند با استفاده از آن خطاهایی که در موقعیت های مختلف روی می دهد را برطرف کند. استثنا ها توسط برنامه به وجود می آیند و شما لازم است آنها را اداره کنید. (مثل خطای تقسیم بر صفر)

دستور try ... catch ... finally ← می توان خطاهای یا استثنا ها را با کمک این دستور مدیریت کرد. کدی که احتمال میدهیم ممکن است در آن خطا رخ دهد را در قسمت try و کدهایی که پس از رخ دادن خطا می خواهیم اجرا شوند را در قسمت catch می نویسیم. بلوک finally شامل کدهایی است که می خواهیم در هر صورت اجرا شوند، چه خطا رخ داده باشد و چه رخ نداده باشد.

با استفاده از کلاس عمومی exception می توان کل خطاهای رخ داده در بلوک try را مدیریت کرد.

**Exp)**

```
int x=50,y=10,z=100;
try
{
    z=x/y;
}
catch(Exception e )
{
    System.out.println(e.getMessage());
}
finally
{
    System.out.println(" z = "+ z);
}
```