

به نام خدا



برنامه‌سازی پیشرفته

دانشگاه شهید بهشتی · دانشکده مهندسی و علوم کامپیوتر

دکتر مجتبی وحیدی اصل

آشنایی با شیعگرایی - بخش دوم

نازنین فروتن

فهرست مطالب

1. انگیزه
2. اشیا و کلاس‌های تغییرناپذیر (Immutable)
3. حوزه متغیرها
4. سطوح دسترسی در جاوا
5. کلمه کلیدی this
6. انتزاع و کپسوله‌سازی
7. طراحی کلاس Loan (وام)
8. طراحی کلاس BMI (شاخص توده بدنی)
9. مثال: کلاس Course (درس)
10. مثال: کلاس StackOfIntegers (پشته اعداد صحیح)

اشیا و کلاس‌های تغییرناپذیر (immutable)

- اگر محتوای یک شیء پس از ایجاد آن نتواند تغییر کند، به آن شیء تغییرناپذیر گفته می‌شود و کلاس مربوطه کلاس تغییرناپذیر نامیده می‌شود.
- اگر شما متد `set` در کلاس `Circle` را در قطعه کد بعدی حذف کنید، کلاس تغییرناپذیر خواهد شد؛ چون `radius` سطح دسترسی `private` دارد و بدون متد `set` قابل دستکاری نمی‌باشد.

کلاسی که همه فیلدهای آن `private` هستند، بدون وجود تغییردهنده‌ها (mutators) لزوماً تغییرناپذیر نمی‌باشد. برای مثال کلاس `Student` که همه فیلدهای آن `private` است و متد تغییردهنده ندارد، همچنان تغییرپذیر می‌باشد.

مثال

```
public class Circle3 {  
    /** The radius of the circle */  
    private double radius = 1;  
  
    /** Construct a circle with a specified radius */  
    public Circle3(double newRadius) {  
        radius = newRadius;  
        numberofObjects++;  
    }  
  
    /** Return radius */  
    public double getRadius() {  
        return radius;  
    }  
  
    /** Set a new radius */  
    public void setRadius(double newRadius) {  
        radius = (newRadius >= 0) ? newRadius : 0;  
    }  
  
    /** Return the area of this circle */  
    public double getArea() {  
        return radius * radius * Math.PI;  
    }  
}
```

مثال

```
public class BirthDate {  
    private int year;  
    private int month;  
    private int day;  
  
    public BirthDate(int newYear, int newMonth, int newDay) {  
        year = newYear;  
        month = newMonth;  
        day = newDay;  
    }  
  
    public void setYear(int newYear) {  
        year = newYear;  
    }  
}
```

```
public class Student {  
    private int id;  
    private BirthDate birthDate;  
  
    public Student(int ssn, int year, int month, int day) {  
        id = ssn;  
        birthDate = new BirthDate(year, month, day);  
    }  
  
    public int getId() {  
        return id;  
    }  
  
    public BirthDate getBirthDate() {  
        return birthDate;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Student student = new Student(111223333, 1970, 5, 3);  
        BirthDate date = student.getBirthDate();  
        date.setYear(2010); // Now the student birth year is changed!  
    }  
}
```

چه کلاسی تغییرناپذیر است؟

برای اینکه کلاس تغییرناپذیر باشد، همه فیلدهای داده‌ای آن باید **private** تعریف شوند و هیچ متدهای تغییردهنده یا متدهای دسترسی وجود نداشته باشد که ارجاعی به یک فیلد داده‌ای قابل تغییر را برگردانند.

ویژگی‌های کلاس تغییرناپذیر

۱. تمام فیلدها (خصوصاً داده‌های مهم) باید **private** و اغلب **final** باشند.

- این باعث می‌شود بیرون از کلاس کسی نتواند آنها را تغییر دهد.

۲. Setter وجود ندارد.

- چون **setter** باعث تغییر وضعیت می‌شود، این کلاس‌ها معمولاً **getter** دارند.

۳. اگر فیلدی خودش **mutable** (مثل لیست یا آرایه) باشد، باید از آن کپی گرفته شود.

- وگرنمی امکان تغییر غیرمستقیم وجود دارد.

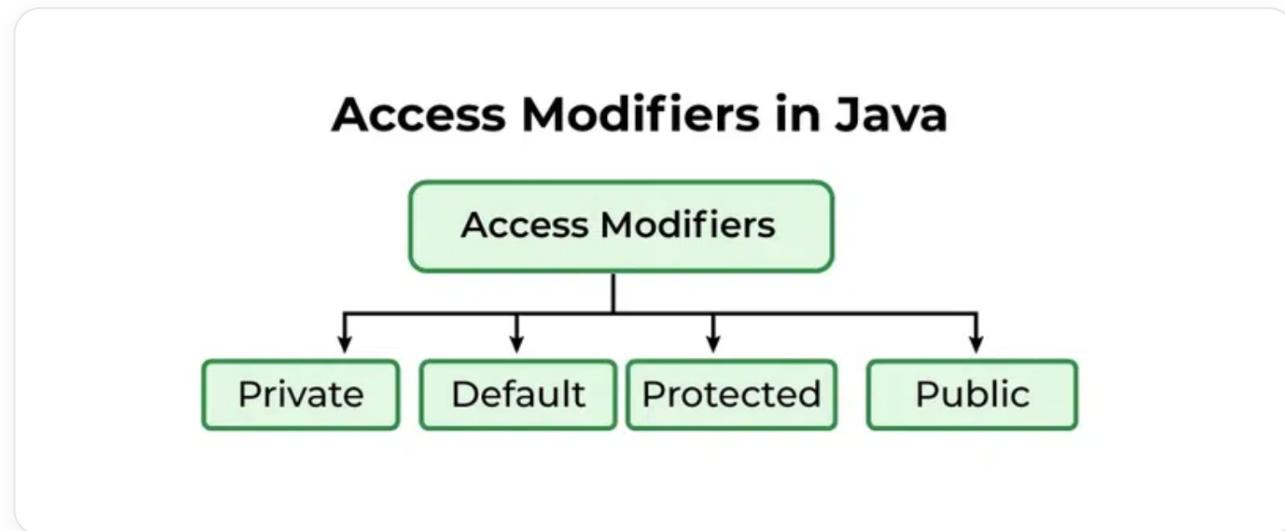
۴. تمام مقداردهی‌ها در **constructor** یا **builder** انجام می‌شود.

حوزه متغیرها

- **متغیرهای نمونه (Instance Variables)** حوزه این متغیرها کل محدوده کلاس است، اما برای دسترسی به آنها باید یک شیء از کلاس ساخته شود. عمرشان برابر با عمر شیء است.
- **متغیرهای استاتیک (Static Variables)** حوزه این متغیرها نیز کل محدوده کلاس است، ولی این متغیرها به همه اشیاء مشترک تعلق دارند. عمرشان تا پایان اجرای برنامه است.
- **متغیرهای محلی (Local Variables)** حوزه آنها از نقطه اعلان تا پایان بلاکی است که در آن تعریف شده‌اند. قبل از استفاده باید مقداردهی شوند.
- **متغیرهای پارامتر (Parameters)** هنگام فراخوانی متدها یا سازنده تعریف می‌شوند و فقط در همان متدها معتبر هستند. پس از پایان اجرای متدها از بین می‌روند.

سطوح دسترسی در جاوا - بررسی دقیق‌تر

- تغییردهنده‌های دسترسی (**access modifiers**) در جاوا دسترسی‌پذیری (حوزه) یک فیلد داده‌ای، متدها، سازنده یا کلاس را مشخص می‌کند.
- در جاوا چهار سطح دسترسی اصلی برای متغیرها، متدها و کلاس‌ها وجود دارد. این سطوح دسترسی میزان قابل مشاهده بودن (**visibility**) و امکان استفاده از آن‌ها در سایر کلاس‌ها یا پکیج‌ها را مشخص می‌کنند:
 - .1: فقط در داخل همان کلاس قابل دسترسی است. **private**
 - .2: در صورت عدم ذکر سطح دسترسی، فقط در همان پکیج قابل دسترسی خواهد بود. **default (package-private)**
 - .3: در همان پکیج و همچنین در کلاس‌های فرزند (**subclass**) حتی در پکیج‌های دیگر قابل دسترسی است. **protected**
 - .4: در همه جا بدون محدودیت قابل دسترسی است. **public**



تغییردهنده دسترسی **private**

اگر سازنده یک کلاس را **private** تعریف کنیم، نمی‌توانیم نمونه‌ای از کلاس را در خارج از آن کلاس ایجاد کنیم.

```
class A{  
    private A(){ } //private constructor  
    void msg(){System.out.println("Hello java");}  
}  
  
public class Simple{  
    public static void main(String args[]){  
        A obj=new A(); //Compile Time Error  
    }  
}
```

تغییردهنده دسترسی default

- اگر شما از هیچ تغییردهنده‌ای استفاده نکنید، با آن فیلد یا متدهای مربوطه تنها از درون پکیجش قابل دسترسی باشد.

```
//save by A.java
package pack;
class A{
    void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;
class B{
    public static void main(String args[]){
        A obj = new A();//Compile Time Error
        obj.msg();//Compile Time Error
    }
}
```

تغییردهنده دسترسی **protected**

- تغییردهنده دسترسی **protected** سبب می‌شود فیلد مربوطه درون پکیج قابل دسترسی باشد یا توسط وراثت در خارج پکیج دسترسی‌پذیر شود.
- این تغییردهنده می‌تواند بر روی فیلد داده‌ای، متدهای سازنده اعمال شود و بر روی کلاس اعمال نمی‌شود.
- در مثال زیر دو پکیج **pack** و **mypack** داریم. کلاس A در پکیج **pack** به صورت **public** تعریف شده، پس در خارج از پکیج قابل دسترسی است. اما متدهای msg در این پکیج به صورت **protected** تعریف شده، در نتیجه در خارج پکیج فقط با وراثت قابل دسترسی می‌باشد:

```
//save by A.java
package pack;
public class A{
    protected void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B extends A{
    public static void main(String args[]){
        B obj = new B();
        obj.msg();
    }
}
```

```
Output:Hello
```

تغییردهنده دسترسی **public**

- در همه جا قابل دسترسی است!
- وسیع‌ترین حوزه دسترسی میان سایر تغییردهنده‌ها را دارد.

```
//save by A.java
```

```
package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
```

```
package mypack;
import pack.*;

class B{
public static void main(String args[]){
A obj = new A();
obj.msg();
}
}
```

```
Output:Hello
```

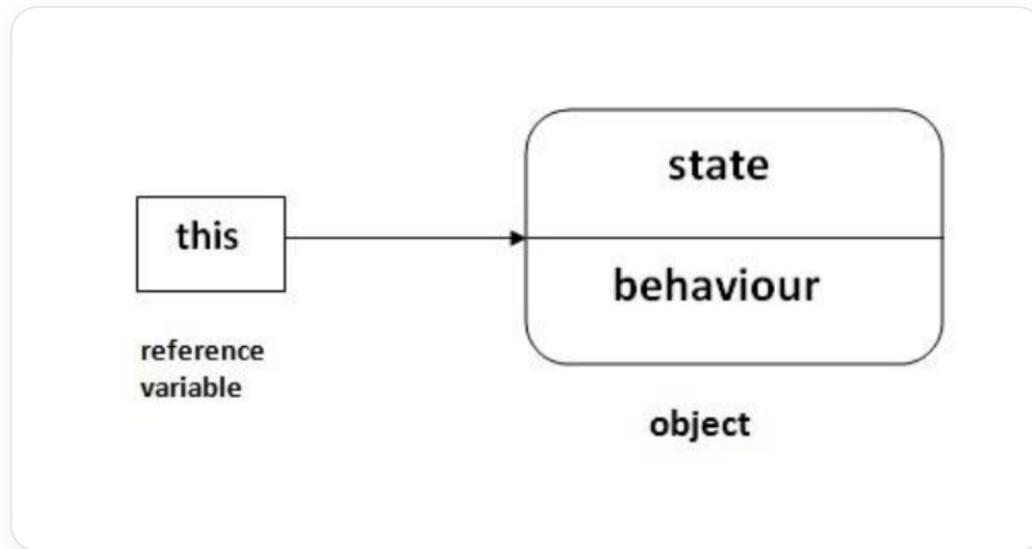
جدول سطوح دسترسی در جاوا

Access Modifier	within class	within package	outside package by subclass only	outside package
Private				
Default				
Protected				
Public				

کلمه کلیدی this

- کلمه کلیدی this نام ارجاعی است که به خود یک شیء اشاره می‌کند.
- کاربرد مهم آن ارجاع به متغیر نمونه کلاس فعلی (جاری) است.
- () This برای فراخوانی سازنده کلاس فعلی استفاده می‌شود.
- برای فراخوانی متدهای کلاس فعلی به طور ضمنی استفاده می‌شود.
- می‌تواند به عنوان آرگومان در فراخوانی یک متدهای استفاده شود.
- می‌تواند به عنوان آرگومان در فراخوانی سازنده ارسال شود.
- به یک سازنده امکان می‌دهد، سازنده دیگری از همان کلاس را فراخوانی نماید.
- برای برگرداندن نمونه کلاس فعلی استفاده می‌شود.

نکته: کلمه کلیدی **this** به نمونه فعلی کلاس اشاره می‌کند.



هرگاه میان متغیر نمونه و پارامتر ارسالی، ابهام وجود داشته باشد، **this** برای رفع ابهام به کار می‌رود.

برای مثال در کد زیر از **this** استفاده نمی‌کنیم و در نتیجه دچار ابهام شدیم:

```
class Student10 {  
    int id;  
    String name;  
  
    Student10(int id, String name) {  
        id = id;  
        name = name;  
    }  
  
    void display() {  
        System.out.println(id + " " + name);  
    }  
  
    public static void main(String args[]) {  
        Student10 s1 = new Student10(111, "Karan");  
        Student10 s2 = new Student10(321, "Aryan");  
        s1.display();  
        s2.display();  
    }  
}
```

ارجاع به متغیر نمونه کلاس فعلی به عنوان فیلد پنهان

در مثال قبلی، پارامتر متد (آرگومان‌های فرمال) و متغیرهای نمونه یکی هستند و به همین دلیل برای تمایز آنها از کلمه **this** استفاده می‌شود.

```
// example of this keyword
class Student11 {
    int id;
    String name;

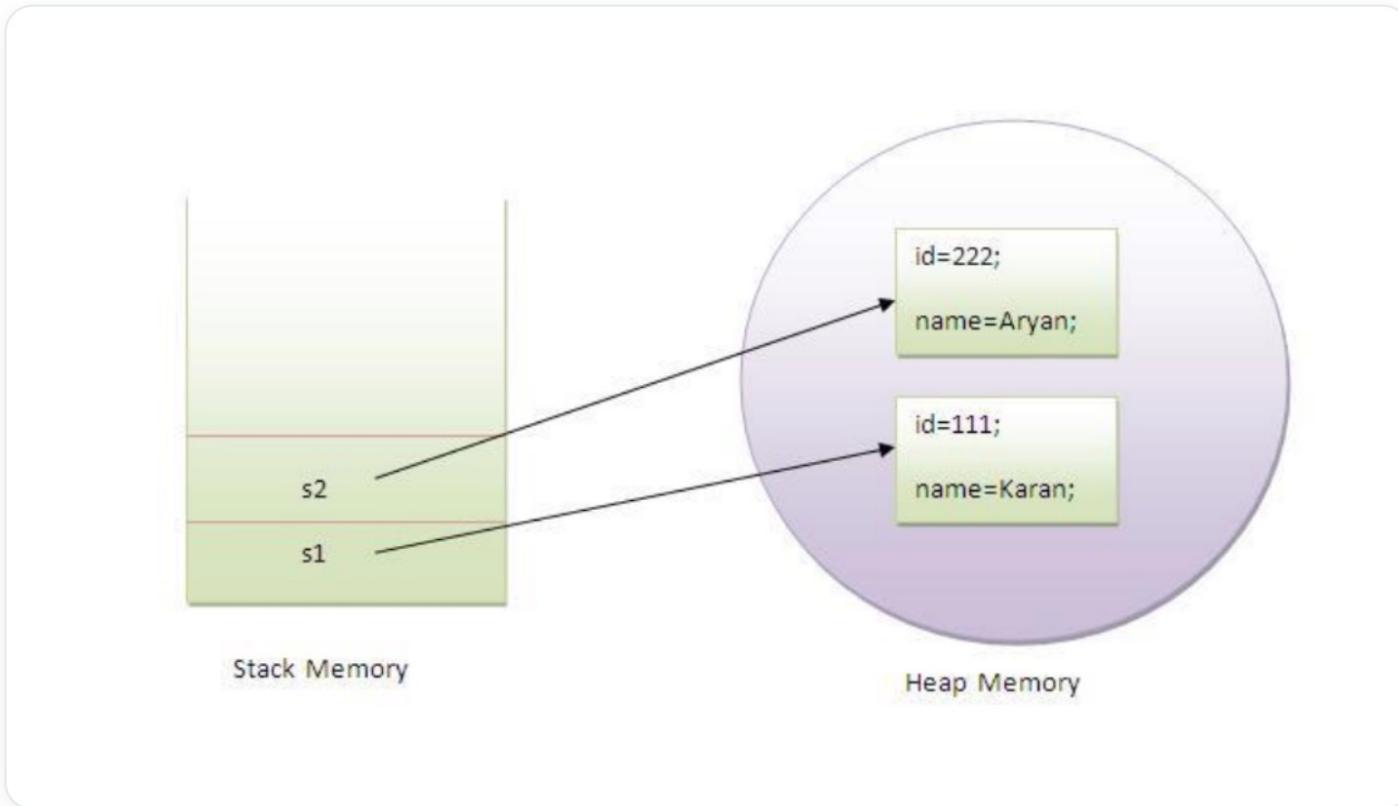
    Student11(int id, String name) {
        this.id = id;
        this.name = name;
    }

    void display() {
        System.out.println(id + " " + name);
    }

    public static void main(String args[]) {
        Student11 s1 = new Student11(111, "Karan");
        Student11 s2 = new Student11(222, "Aryan");
        s1.display();
        s2.display();
    }
}
```

```
111 Karan null
222 Aryan delhi
```

نکته: کلمه کلیدی **this** به فیلدهای داده‌ای اشیای کلاس جاری اشاره می‌کند.



نکته: اگر نام متغیرهای نمونه و پارامترهای متد کلاس متفاوت باشند، در این صورت به **this** نیازی نداریم.

```
class Student12 {  
    int id;  
    String name;  
  
    Student12(int i, String n) {  
        id = i;  
        name = n;  
    }  
  
    void display() {  
        System.out.println(id + " " + name);  
    }  
  
    public static void main(String args[]) {  
        Student12 e1 = new Student12(111, "Karan");  
        Student12 e2 = new Student12(222, "Aryan");  
        e1.display();  
        e2.display();  
    }  
}
```

```
111 Karan null  
222 Aryan delhi
```

نکته: فراخوانی سازنده `(this)` می‌تواند برای فراخوانی سازنده کلاس فعلی (زنجیره سازنده‌ها) استفاده شود.

```
class Student13 {  
    int id;  
    String name;  
  
    Student13() {  
        System.out.println("default constructor is invoked");  
    }  
  
    Student13(int id, String name) {  
        this(); // it is used to invoke current class constructor.  
        this.id = id;  
        this.name = name;  
    }  
  
    void display() {  
        System.out.println(id + " " + name);  
    }  
  
    public static void main(String args[]) {  
        Student13 e1 = new Student13(111, "Karan");  
        Student13 e2 = new Student13(222, "Aryan");  
        e1.display();  
        e2.display();  
    }  
}
```

```
default constructor is invoked  
default constructor is invoked  
111 Karan  
222 Aryan
```

چه موقع از فراخوانی سازنده `this()` استفاده کنیم؟

- فراخوانی سازنده `this()` در موقع استفاده مجدد از یک سازنده در سازنده دیگر استفاده می‌شود.
- این کلمه زنجیره‌ای را میان سازنده‌ها برقرار می‌کند.

```
class Student14 {  
    int id;  
    String name;  
    String city;  
  
    Student14(int id, String name) {  
        this.id = id;  
        this.name = name;  
    }  
  
    Student14(int id, String name, String city) {  
        this(id, name); // now no need to initialize id and name  
        this.city = city;  
    }  
  
    void display() {  
        System.out.println(id + " " + name + " " + city);  
    }  
  
    public static void main(String args[]) {  
        Student14 e1 = new Student14(111, "Karan");  
        Student14 e2 = new Student14(222, "Aryan", "delhi");  
        e1.display();  
        e2.display();  
    }  
}
```

```
111 Karan null  
222 Aryan delhi
```

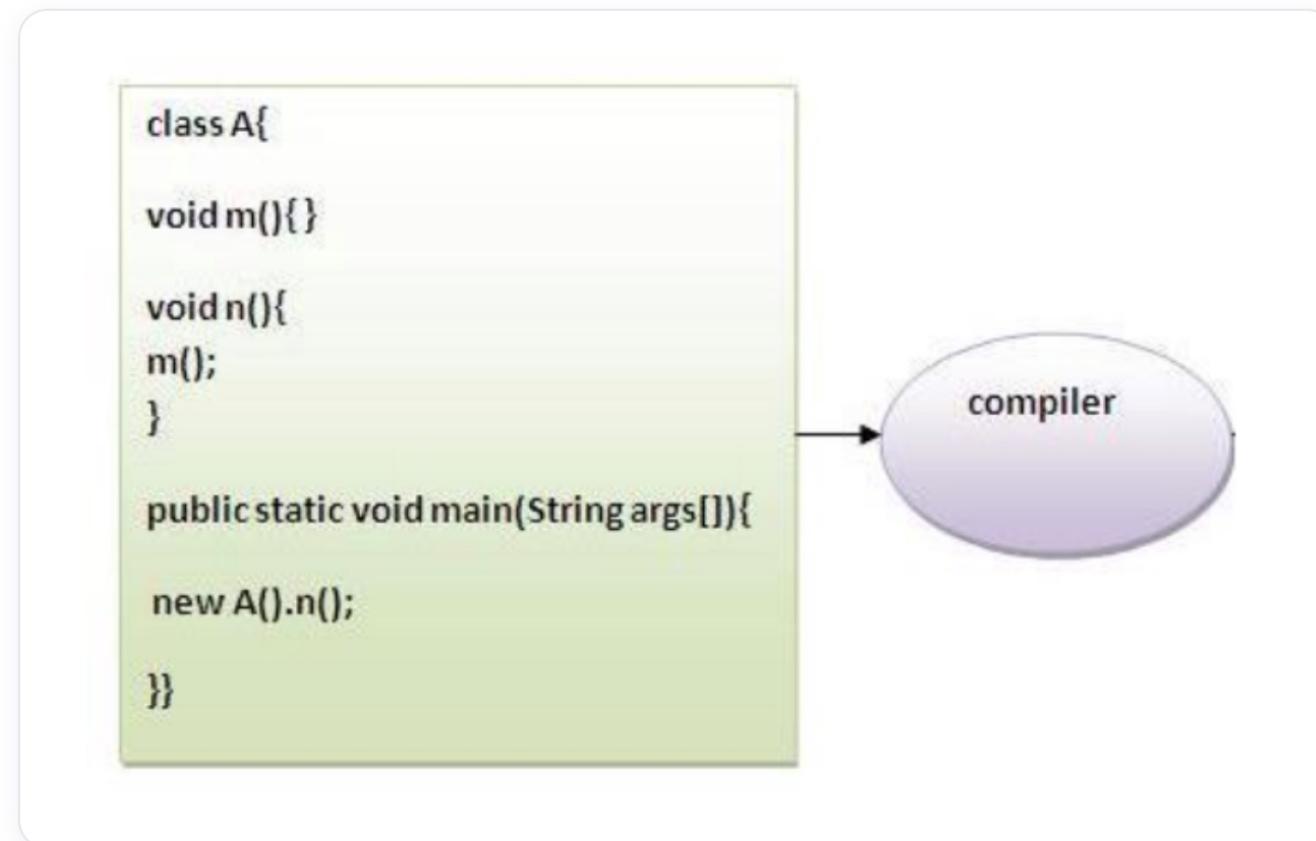
یک قاعده: فراخوانی `this()` باید اولین جمله در سازنده باشد.

```
class Student15 {  
    int id;  
    String name;  
  
    Student15() {  
        System.out.println("default constructor is invoked");  
    }  
  
    Student15(int id, String name) {  
        id = id;  
        name = name;  
        this(); // must be the first statement  
    }  
  
    void display() {  
        System.out.println(id + " " + name);  
    }  
  
    public static void main(String args[]) {  
        Student15 e1 = new Student15(111, "Karan");  
        Student15 e2 = new Student15(222, "Aryan");  
        e1.display();  
        e2.display();  
    }  
}
```

| `this(); // must be the first statement`
call to this must be first statement in constructor

کلمه this می‌تواند برای فراخوانی متدهای کلاس فعلی (به طور ضمنی) استفاده شود.

- شما می‌توانید متدهای کلاس جاری (فعلی) با **this** فراخوانی کنید.
- اگر شما از این کلمه استفاده نکنید، کامپایلر به طور خودکار این کلمه را در هنگام فراخوانی متدهای اضافه می‌کند. برای مثال:



```
class S {  
    void m() {  
        System.out.println("method is invoked");  
    }  
  
    void n() {  
        this.m(); // no need because compiler does it for you.  
    }  
  
    void p() {  
        n(); // compiler will add this to invoke n() method as this.n()  
    }  
  
    public static void main(String args[]) {  
        S s1 = new S();  
        s1.p();  
    }  
}
```

method is invoked

آیا this دقیقا همان ریموت کنترل شیء است؟

- برای اثبات اینکه this به متغیر نمونه کلاس قعلی اشاره می‌کند، برنامه زیر را ملاحظه کنید:

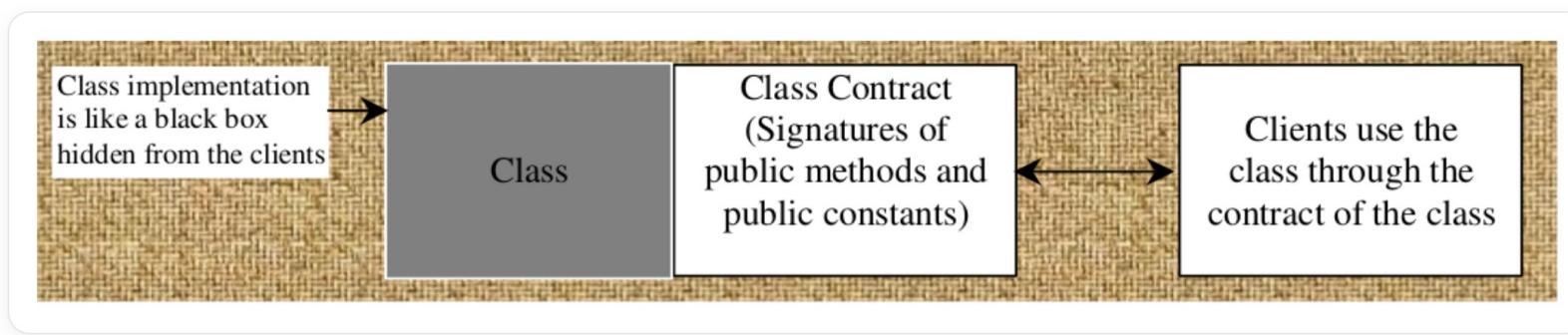
```
class A5 {  
    void m() {  
        System.out.println(this); // prints same reference ID  
    }  
  
    public static void main(String args[]) {  
        A5 obj = new A5();  
        System.out.println(obj); // prints the reference ID  
  
        obj.m();  
    }  
}
```

REPL.\$JShell\$13\$A5@4b45c47b

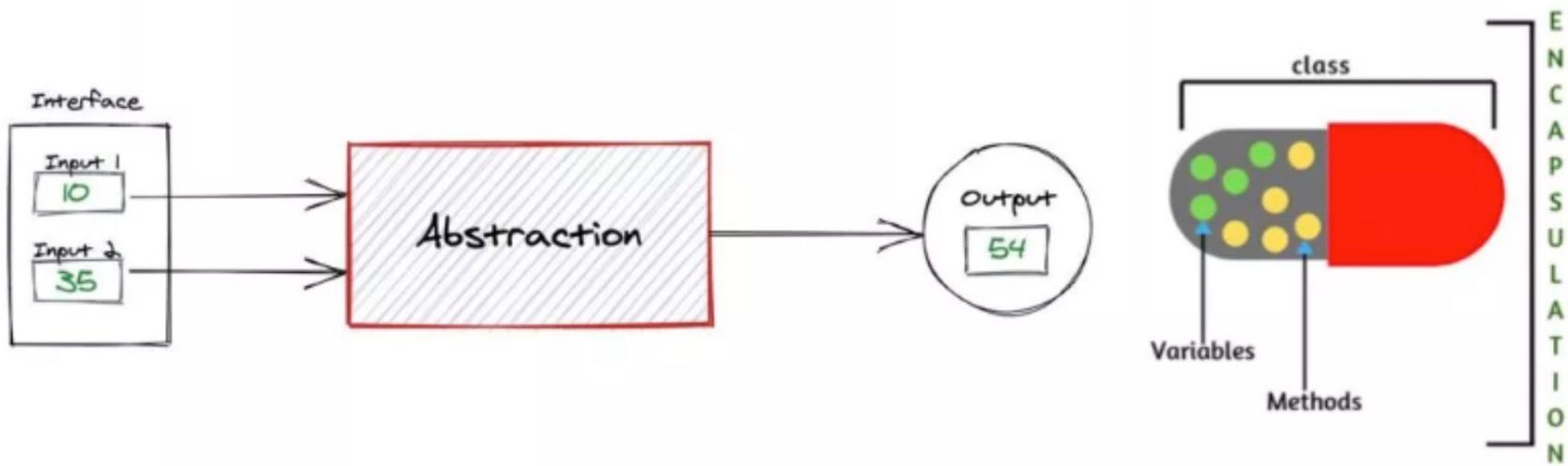
REPL.\$JShell\$13\$A5@4b45c47b

انتزاع و بسته‌بندی کلاس

- انتزاع کلاس به معنای تفکیک پیاده‌سازی کلاس از استفاده آن کلاس است.
- ایجادکننده کلاس توصیفی از کلاس را ارائه می‌کند و به کاربر می‌گوید چگونه از کلاس استفاده کند.
- کاربر کلاس نیاز نیست بداند کلاس چگونه نوشته شده است. جزئیات پیاده‌سازی بسته‌بندی (کپسوله‌سازی) شده و از دید کاربر پنهان است.
- **انتزاع (Abstraction):** یعنی فقط بخش‌های ضروری از یک کلاس یا شیء در اختیار کاربر قرار گیرد و جزئیات غیرضروری پنهان شوند. این کار معمولاً با استفاده از **abstract class** و **interface** انجام می‌شود.
- **بسته‌بندی (Encapsulation):** یعنی داده‌ها (متغیرها) و رفتارها (متدها) در یک واحد (کلاس) قرار بگیرند و دسترسی به آنها با سطح دسترسی مثل **private**, **public**, **protected** کنترل شود.
- انتزاع می‌پرسد «چه کاری انجام می‌شود؟» اما بسته‌بندی می‌پرسد «چگونه و توسط چه کسی قابل دسترسی است؟».



Abstraction Vs Encapsulation In Java



مقایسه انتزاع و محصورسازی

محصورسازی (Encapsulation)	انتزاع (Abstraction)	ویژگی
پنهان کردن داده‌ها و کنترل دسترسی به آن‌ها	پنهان کردن جزئیات غیرضروری و نمایش فقط عملکرد	تعریف
«چگونه داده‌ها محافظت و مدیریت می‌شوند؟»	«چه کاری انجام می‌شود؟»	تمرکز
با سطوح دسترسی (private, public, protected)	با کلاس‌های انتزاعی و اینترفیس‌ها	پیاده‌سازی
جزئیات داخلی موتور و داده‌ها پنهان هستند	رانده فقط می‌بینند ماشین چگونه رانده می‌شود	مثال

طراحی کلاس وام

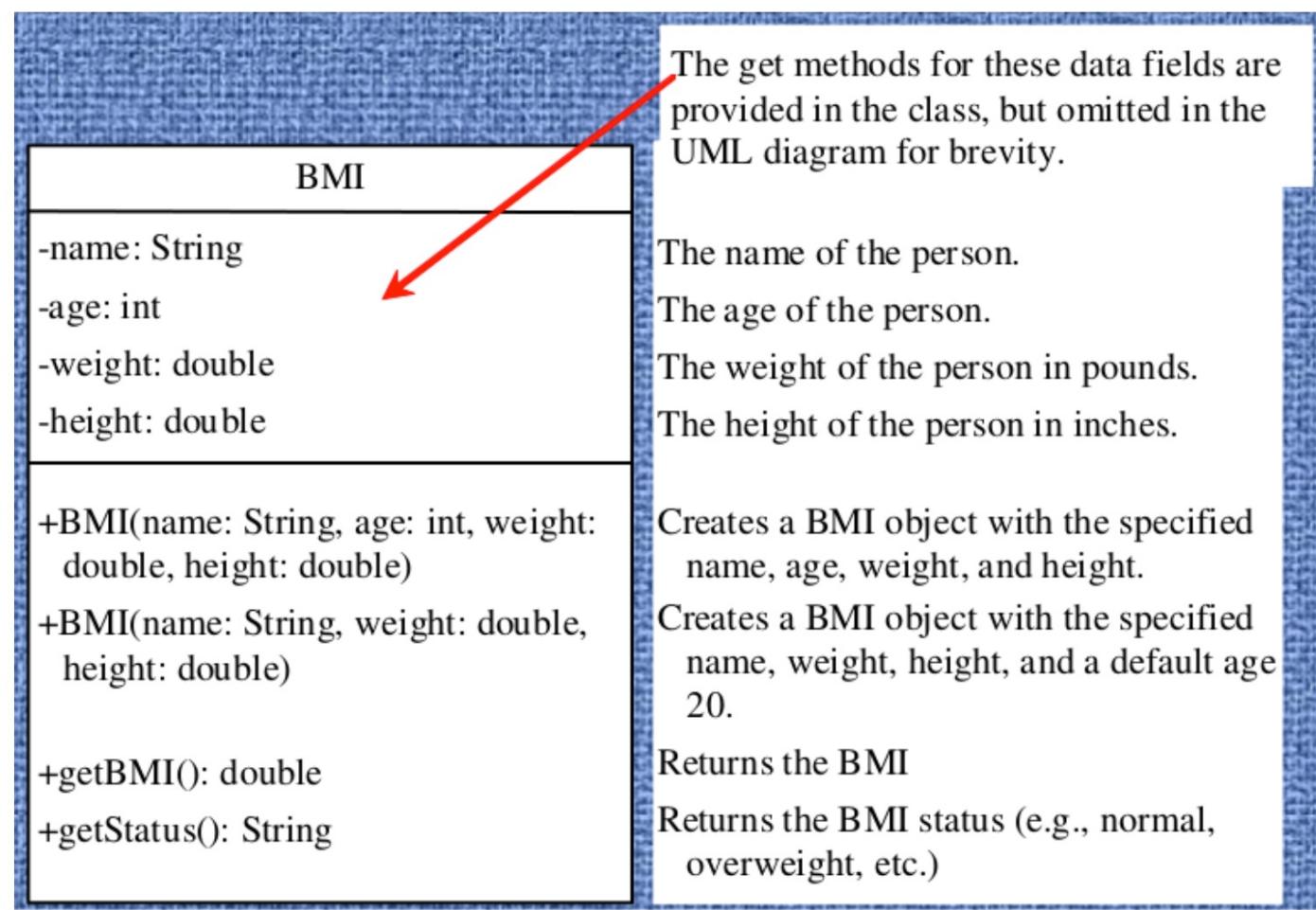
Loan	
-annualInterestRate: double	The annual interest rate of the loan (default: 2.5).
-numberOfYears: int	The number of years for the loan (default: 1)
-loanAmount: double	The loan amount (default: 1000).
-loanDate: Date	The date this loan was created.
+Loan()	Constructs a default Loan object.
+Loan(annualInterestRate: double, numberOfYears: int, loanAmount: double)	Constructs a loan with specified interest rate, years, and loan amount.
+getAnnualInterestRate(): double	Returns the annual interest rate of this loan.
+getNumberOfYears(): int	Returns the number of the years of this loan.
+getLoanAmount(): double	Returns the amount of this loan.
+getLoanDate(): Date	Returns the date of the creation of this loan.
+setAnnualInterestRate(annualInterestRate: double): void	Sets a new annual interest rate to this loan.
+setNumberOfYears(numberOfYears: int): void	Sets a new number of years to this loan.
+setLoanAmount(loanAmount: double): void	Sets a new amount to this loan.
+getMonthlyPayment(): double	Returns the monthly payment of this loan.
+getTotalPayment(): double	Returns the total payment of this loan.

```
public class Loan {  
    private double annualInterestRate;  
    private int numberOfYears;  
    private double loanAmount;  
    private java.util.Date loanDate;  
  
    /** Default constructor */  
    public Loan() {  
        this(2.5, 1, 1000);  
    }  
  
    /** Construct a loan with specified annual interest rate,  
     * number of years and loan amount */  
    public Loan(double annualInterestRate, int numberOfYears, double loanAmount) {  
        this.annualInterestRate = annualInterestRate;  
        this.numberOfYears = numberOfYears;  
        this.loanAmount = loanAmount;  
        loanDate = new java.util.Date();  
    }  
  
    /** Return annualInterestRate */  
    public double getAnnualInterestRate() {  
        return annualInterestRate;  
    }  
}
```

```
/** Set a new annualInterestRate */  
public void setAnnualInterestRate(double annualInterestRate) {  
    this.annualInterestRate = annualInterestRate;  
}  
  
/** Return numberOfYears */  
public int getNumberOfYears() {  
    return numberOfYears;  
}  
  
/** Set a new numberOfYears */  
public void setNumberOfYears(int numberOfYears) {  
    this.numberOfYears = numberOfYears;  
}  
  
/** Return loanAmount */  
public double getLoanAmount() {  
    return loanAmount;  
}  
  
/** Set a new loanAmount */  
public void setLoanAmount(double loanAmount) {  
    this.loanAmount = loanAmount;  
}
```

```
/** Find monthly payment */  
public double getMonthlyPayment() {  
    double monthlyInterestRate = annualInterestRate / 1200;  
    double monthlyPayment = loanAmount * monthlyInterestRate /  
        (1 - (Math.pow(1 / (1 + monthlyInterestRate), numberOfYears * 12)));  
    return monthlyPayment;  
}  
  
/** Find total payment */  
public double getTotalPayment() {  
    double totalPayment = getMonthlyPayment() * numberOfYears * 12;  
    return totalPayment;  
}  
  
/** Return loan date */  
public java.util.Date getLoanDate() {  
    return loanDate;  
}
```

طراحی کلاس BMI (Body Mass Index)



```
public class BMI {  
    private String name;  
    private int age;  
    private double weight; // in pounds  
    private double height; // in inches  
    public static final double KILOGRAMS_PER_POUND = 0.45359237;  
    public static final double METERS_PER_INCH = 0.0254;  
  
    public BMI(String name, int age, double weight, double height) {  
        this.name = name;  
        this.age = age;  
        this.weight = weight;  
        this.height = height;  
    }  
  
    public BMI(String name, double weight, double height) {  
        this(name, 20, weight, height);  
    }  
  
    public double getBMI() {  
        double bmi = weight * KILOGRAMS_PER_POUND /  
            ((height * METERS_PER_INCH) * (height * METERS_PER_INCH));  
        return Math.round(bmi * 100) / 100.0;  
    }  
}
```

```
public String getStatus() {  
    double bmi = getBMI();  
    if (bmi < 16)  
        return "seriously underweight";  
    else if (bmi < 18)  
        return "underweight";  
    else if (bmi < 24)  
        return "normal weight";  
    else if (bmi < 29)  
        return "over weight";  
    else if (bmi < 35)  
        return "seriously over weight";  
    else  
        return "gravely over weight";  
}  
  
public String getName() {  
    return name;  
}  
  
public int getAge() {  
    return age;  
}  
  
public double getWeight() {  
    return weight;  
}
```

طراحی کلاس Course

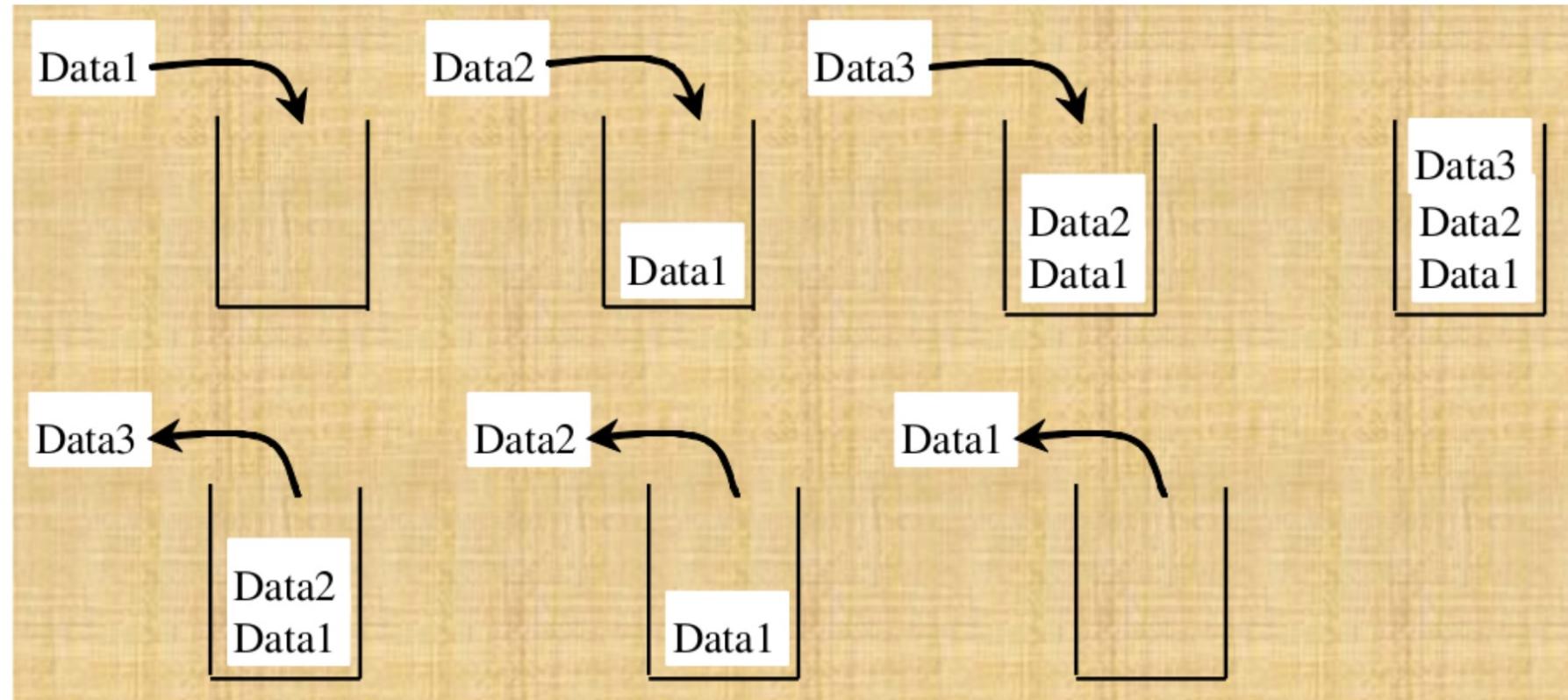
Course	
-name: String	The name of the course.
-students: String[]	The students who take the course.
-numberOfStudents: int	The number of students (default: 0).
+Course(name: String)	Creates a Course with the specified name.
+getName(): String	Returns the course name.
+addStudent(student: String): void	Adds a new student to the course list.
+getStudents(): String[]	Returns the students for the course.
+getNumberOfStudents(): int	Returns the number of students for the course.

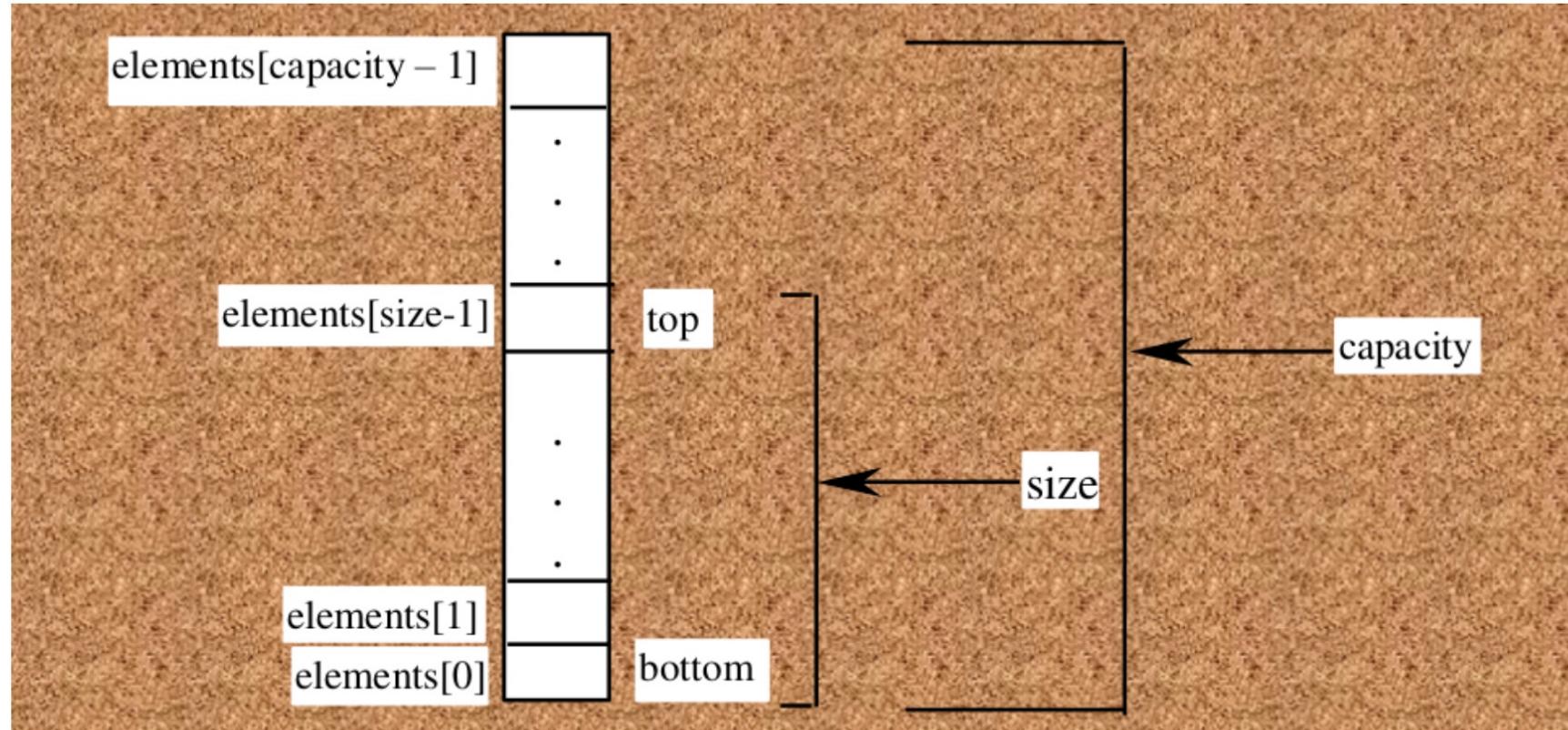
```
public class Course {  
    private String courseName;  
    private String[] students = new String[100];  
    private int numberOfStudents;  
  
    public Course(String courseName) {  
        this.courseName = courseName;  
    }  
  
    public void addStudent(String student) {  
        students[numberOfStudents] = student;  
        numberOfStudents++;  
    }  
  
    public String[] getStudents() {  
        return students;  
    }  
  
    public int getNumberOfStudents() {  
        return numberOfStudents;  
    }  
  
    public String getCourseName() {  
        return courseName;  
    }  
}
```

طراحی کلاس StackOfIntegers

StackOfIntegers	
-elements: int[]	An array to store integers in the stack.
-size: int	The number of integers in the stack.
+StackOfIntegers()	Constructs an empty stack with a default capacity of 16.
+StackOfIntegers(capacity: int)	Constructs an empty stack with a specified capacity.
+empty(): boolean	Returns true if the stack is empty.
+peek(): int	Returns the integer at the top of the stack without removing it from the stack.
+push(value: int): int	Stores an integer into the top of the stack.
+pop(): int	Removes the integer at the top of the stack and returns it.
+getSize(): int	Returns the number of elements in the stack.

شماتیک و نحوه کارکرد:





```
public class StackOfIntegers {  
    private int[] elements;  
    private int size;  
    public static final int DEFAULT_CAPACITY = 16;  
  
    /** Construct a stack with the default capacity 16 */  
    public StackOfIntegers() {  
        this(DEFAULT_CAPACITY);  
    }  
  
    /** Construct a stack with the specified maximum capacity */  
    public StackOfIntegers(int capacity) {  
        elements = new int[capacity];  
    }  
  
    /** Push a new integer into the top of the stack */  
    public void push(int value) {  
        if (size >= elements.length) {  
            int[] temp = new int[elements.length * 2];  
            System.arraycopy(elements, 0, temp, 0, elements.length);  
            elements = temp;  
        }  
        elements[size++] = value;  
    }  
}
```

```
    /** Return and remove the top element from the stack */  
    public int pop() {  
        return elements[--size];  
    }  
  
    /** Return the top element from the stack */  
    public int peek() {  
        return elements[size - 1];  
    }  
  
    /** Test whether the stack is empty */  
    public boolean empty() {  
        return size == 0;  
    }  
  
    /** Return the number of elements in the stack */  
    public int getSize() {  
        return size;  
    }  
}
```

خودمون رو بسنجیم

این بخش برای این طراحی شده که در پایان مطالعه این اسلاید، بتونی خودت رو محک بزنی و ببینی آیا مفاهیم رو به خوبی یاد گرفتی یا نه. سوالات زیر رو مرور کن و سعی کن بدون نگاه کردن به متن درس، به اون ها پاسخ بدی.

- چرا برای تعریف ویژگی‌ها (fields) معمولاً از **private** استفاده می‌کنیم؟
- چطور با استفاده از **کپسوله‌سازی (Encapsulation)** می‌توانیم امنیت داده‌ها را حفظ کنیم؟
- تفاوت اصلی بین **انتزاع (Abstraction)** و **محصورسازی (Encapsulation)** چیه؟

پایان

در صورت هرگونه سوال یا پیشنهاد می‌توانید با من
در ارتباط باشید:

gmail: foroutanazanin@gmail.com
telegram: @naforoutan