

به نام خدا



برنامه‌سازی ییشوفته

دانشگاه شهید بهشتی · دانشکده مهندسی کامپیووتر

دکتر مجتبی وحیدی اصل

آشنایی با شیء‌گرایی - بخش دوم

نازین فروتن

فهرست مطالب

1. انگیزه
2. اشیا و کلاس‌های تغییرنپذیر (Immutable)
3. حوزه متغیرها
4. سطوح دسترسی در جاوا
 - private •
 - default •
 - protected •
 - public •
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)

انگیزه

در جلسه قبل با برخی مزایای برنامه نویسی شءگرا آشنا شدیم.

در این جلسه برخی دیگر از این مزایا را با ارائه مثال بررسی خواهیم نمود

اشیا و کلاس‌های تغییرنپذیر (immutable)

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

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

```
In [ ]: /* مثالی از کلاس تغییر پذیر */

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;
        number0f0bjects++;
    }

    /** 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;
    }
}
```

In []: /* مثالی از کلاس تغییر ناپذیر */

```
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;
        number0fObjects++;
    }

    /** Return radius */
    public double getRadius() {
        return radius;
    }

    /** Set a new radius */
    private void setRadius(double newRadius) {
        radius = (newRadius >= 0) ? newRadius : 0;
    }

    /** Return the area of this circle */
    public double getArea() {
        return radius * radius * Math.PI;
    }
}
```

```
    }  
}
```

مثال

```
In [14]: 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;  
    }  
}
```

```
In [15]: 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;
}
```

```
In [16]: 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!
    }
}

Test.main(new String[] {});
```

چه کلاسی تغییرناپذیر است؟

تعریف شوند و هیچ متدهای تغییردهنده یا متدهای دسترسی وجود نداشته باشد، همه فیلدات داده‌ای آن باید یک فیلد داده‌ای قابل تغییر را برگردانند.

حوزه متغیرها

- حوزه متغیرهای نمونه و استاتیک کل محدوده کلاس است. این متغیرها می‌توانند در هر جایی داخل کلاس تعریف شوند.
- حوزه یک متغیر محلی از نقطه اعلان آن آغاز و تا انتهای بلاکی که حاوی آن متغیر است ادامه می‌یابد. یک متغیر محلی باید پیش از استفاده مقداردهی شده باشد.

سطوح دسترسی در جاوا - بررسی دقیق‌تر

- در جاوا دسترسی‌پذیری (حوزه) یک فیلد داده‌ای، متدها، سازنده یا کلاس را مشخص می‌کند (access modifiers) تغییردهنده‌های دسترسی و امکان (visibility) در جاوا چهار سطح دسترسی اصلی برای متغیرها، متدها و کلاس‌ها وجود دارد. این سطوح دسترسی میزان قابل مشاهده بودن استفاده از آن‌ها در سایر کلاس‌ها یا پکیج‌ها را مشخص می‌کنند:

1. **private**: فقط در داخل همان کلاس قابل دسترسی است.

2. **default (package-private)**: در صورت عدم ذکر سطح دسترسی، فقط در همان پکیج قابل دسترسی خواهد بود.

3. **protected**: حتی در پکیج‌های دیگر قابل دسترسی است (subclass) در همان پکیج و همچنین در کلاس‌های فرزند.

4. **public**: در همه جا بدون محدودیت قابل دسترسی است.

تغییردهنده دسترسی private

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

```
In [ ]: 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  
    }  
}
```

```
}
```

```
A.main(new String[] {});
```

تغییردهنده دسترسی default

- اگر شما از هیچ تغییردهنده‌ای استفاده نکنید، با آن فیلد یا متدهای طور پیش‌فرض default بروخود می‌شود. این تغییردهنده سبب می‌شود فیلد یا متدهای مربوطه تنها از درون پکیج قابل دسترسی باشد.
- در مثال زیر دو پکیج pack و mypack داریم:

```
In [ ]:
```

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

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

B.main(new String[]{});
```

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، متدها یا 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:

```
In [ ]: public class A {
    protected void msg() {
        System.out.println("Hello");
    }
}

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

B.main(new String[]{});
```

تغییردهنده دسترسی `public`

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

```
In [ ]: // save by A.java
public class A {
```

```

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

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

B.main(new String[]{});

```

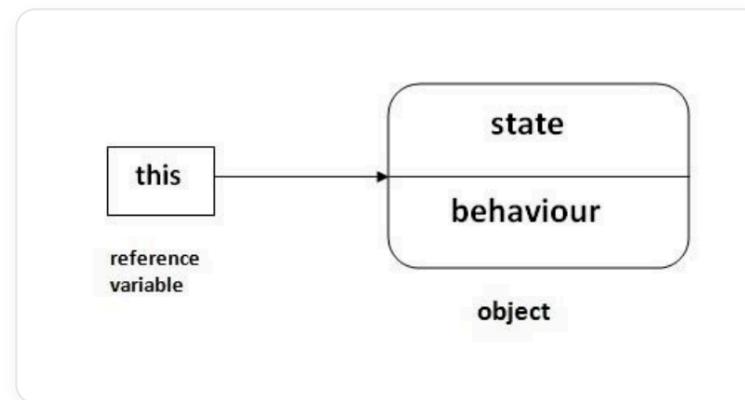
جدول سطوح دسترسی در جاوا

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

کلمه کلیدی this

- نام ارجاعی است که به خود یک شیء اشاره می‌کند **this** کلمه کلیدی.
- کاربرد مهم آن ارجاع به متغیر نمونه کلاس فعلی (جاری) است.
- برای فراخوانی سازنده کلاس فعلی استفاده می‌شود `()` (`This`)
- برای فراخوانی متد کلاس فعلی به طور ضمنی استفاده می‌شود.
- می‌تواند به عنوان آرگومان در فراخوانی یک متد 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.

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



استفاده نمی‌کنیم و در نتیجه this برای مثال در کد زیر از برای رفع ابهام به کار می‌رود، هرگاه میان متغیر نمونه و پارامتر ارسالی، ابهام وجود داشته باشد دچار ابهام شدیم:

```
In [ ]: 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.

```
In [ ]: // example of this keyword
class Student11 {
    int id;
    String name;

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

```
    this.name = name;
}

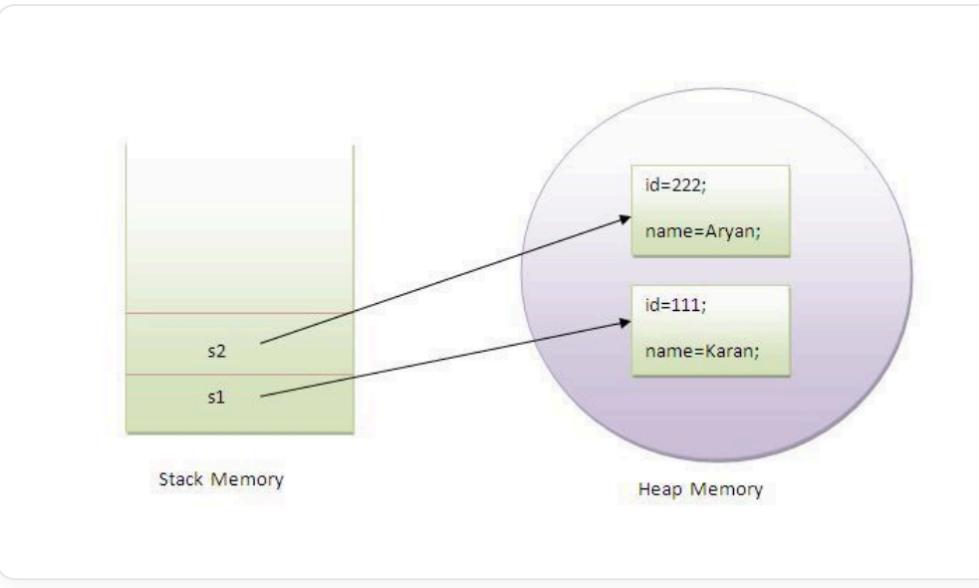
void display() {
    System.out.println(id + " " + name);
}

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

Student11.main(new String[]{});

```

نکته: کلمه کلیدی this به فیلد های داده ای اشیای کلاس جاری اشاره می کند.



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

```
In [ ]: 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();
    }
}

Student12.main(new String[]{});
```

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

```
In [ ]: // Program of this() constructor call (constructor chaining)
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();
    }
}
```

```
Student13.main(new String[]{});
```

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.

```
In [ ]: 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();  
    }  
}
```

```
}
```

```
Student14.main(new String[]{});
```

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

```
In [ ]:
```

```
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
    }

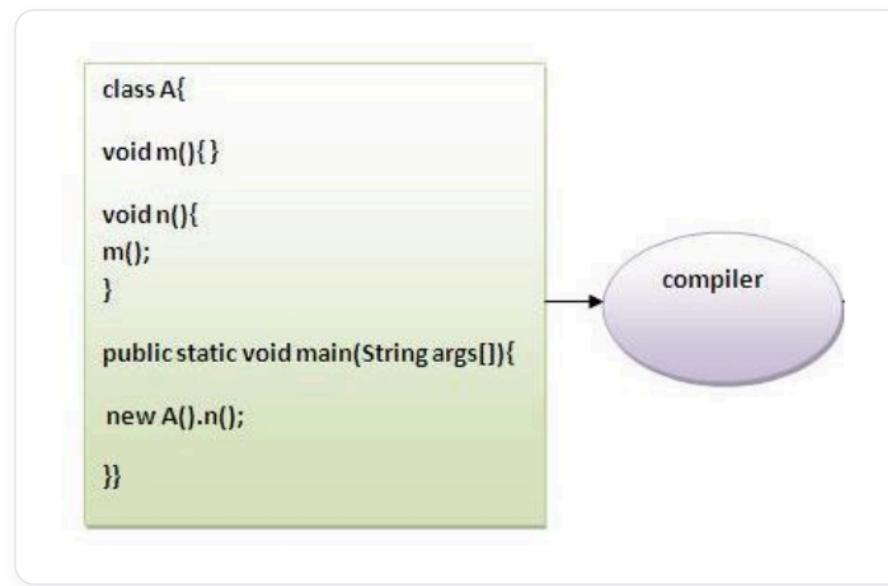
    void display() {
        System.out.println(id + " " + name);
    }

    public static void main(String args[]) {
        Student15 e1 = new Student15(111, "Karan");
        Student15 e2 = new Student15(222, "Aryan");
        e1.display();
        e2.display();
    }
}

Student15.main(new String[]{});
```

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

- شما می‌توانید متدهای از 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:



```
In [ ]: // یک مثال دیگر

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();
}
}

S.main(new String[]{});

```

آیا **this** دقیقا همان ریموت کنترل شیء است؟

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

```

In [ ]: 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

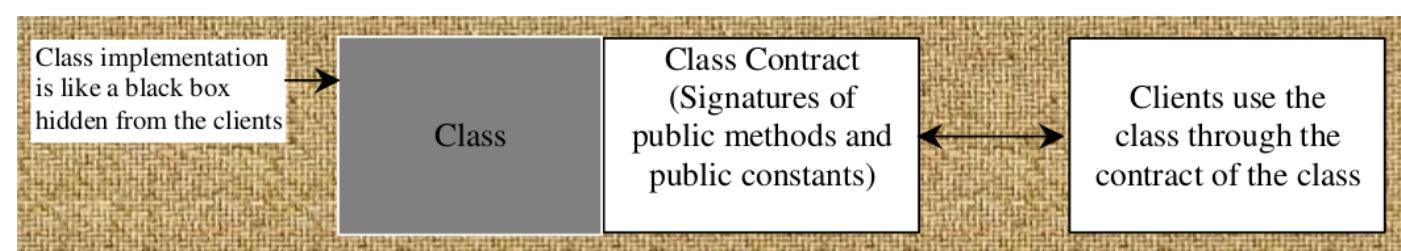
        obj.m();
    }
}

A5.main(new String[]{});

```

انتزاع و بسته‌بندی کلاس

- انتزاع کلاس به معنای تفکیک پیاده‌سازی کلاس از استفاده آن کلاس است.
- ایجادکننده کلاس توصیف (توضیحاتی) از کلاس را ارائه می‌کند و به کاربر می‌گوید چگونه از کلاس استفاده کند.
- کاربر کلاس نیاز نیست بداند کلاس چگونه نوشته شده است (پیاده‌سازی شده است). جزئیات پیاده‌سازی بسته‌بندی (کپسوله‌بندی) شده از دید کاربر پنهان است.



طراحی کلاس و ام

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.

```
In [3]: 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), number0fYears * 12)));
    return monthlyPayment;
}

/** Find total payment */
public double getTotalPayment() {
    double totalPayment = getMonthlyPayment() * number0fYears * 12;
    return totalPayment;
}

/** Return loan date */
public java.util.Date getLoanDate() {
    return loanDate;
}
}
```

```
In [ ]: import java.util.Scanner;

public class TestLoanClass {
    /** Main method */
    public static void main(String[] args) {

        double annualInterestRate = 8.25;
        int number0fYears = 5;
        double loanAmount = 120000.95;

        /*
        // Create a Scanner
        Scanner input = new Scanner(System.in);

        // Enter yearly interest rate
        System.out.print(
            "Enter yearly interest rate, for example, 8.25: ");
        double annualInterestRate = input.nextDouble();

        // Enter number of years
        System.out.print("Enter number of years as an integer: ");
        int number0fYears = input.nextInt();
    }
}
```

```
// Enter loan amount
System.out.print("Enter loan amount, for example, 120000.95: ");
double loanAmount = input.nextDouble();
*/
// Create Loan object
Loan loan = new Loan(annualInterestRate, numberOfYears, loanAmount);

// Display loan date, monthly payment, and total payment
System.out.printf("The loan was created on %s%n" +
    "The monthly payment is %.2f%nThe total payment is %.2f%n",
    loan.getLoanDate().toString(), loan.getMonthlyPayment(),
    loan.getTotalPayment());
}

TestLoanClass.main(new String[]{});
```

طراحی کلاس BMI (Body Mass Index)

BMI	
-name: String	The name of the person.
-age: int	The age of the person.
-weight: double	The weight of the person in pounds.
-height: double	The height of the person in inches.
+BMI(name: String, age: int, weight: double, height: double)	Creates a BMI object with the specified name, age, weight, and height.
+BMI(name: String, weight: double, height: double)	Creates a BMI object with the specified name, weight, height, and a default age 20.
+getBMI(): double	Returns the BMI
+getStatus(): String	Returns the BMI status (e.g., normal, overweight, etc.)

```
In [6]: 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;
    }

    public double getHeight() {
        return height;
    }
```

```
    }  
}
```

```
In [ ]: public class UseBMIClass {  
    public static void main(String[] args) {  
        BMI bmi1 = new BMI("John Doe", 18, 145, 70);  
        System.out.println("The BMI for " + bmi1.getName() + " is "  
            + bmi1.getBMI() + " " + bmi1.getStatus());  
  
        BMI bmi2 = new BMI("Peter King", 215, 70);  
        System.out.println("The BMI for " + bmi2.getName() + " is "  
            + bmi2.getBMI() + " " + bmi2.getStatus());  
    }  
}  
  
UseBMIClass.main(new String[]{});
```

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.

```
In [8]: 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;
    }
}
```

```
In [9]: public class TestCourse {
    public static void main(String[] args) {
        Course course1 = new Course("Data Structures");
        Course course2 = new Course("Database Systems");

        course1.addStudent("Peter Jones");
        course1.addStudent("Brian Smith");
        course1.addStudent("Anne Kennedy");

        course2.addStudent("Peter Jones");
        course2.addStudent("Steve Smith");

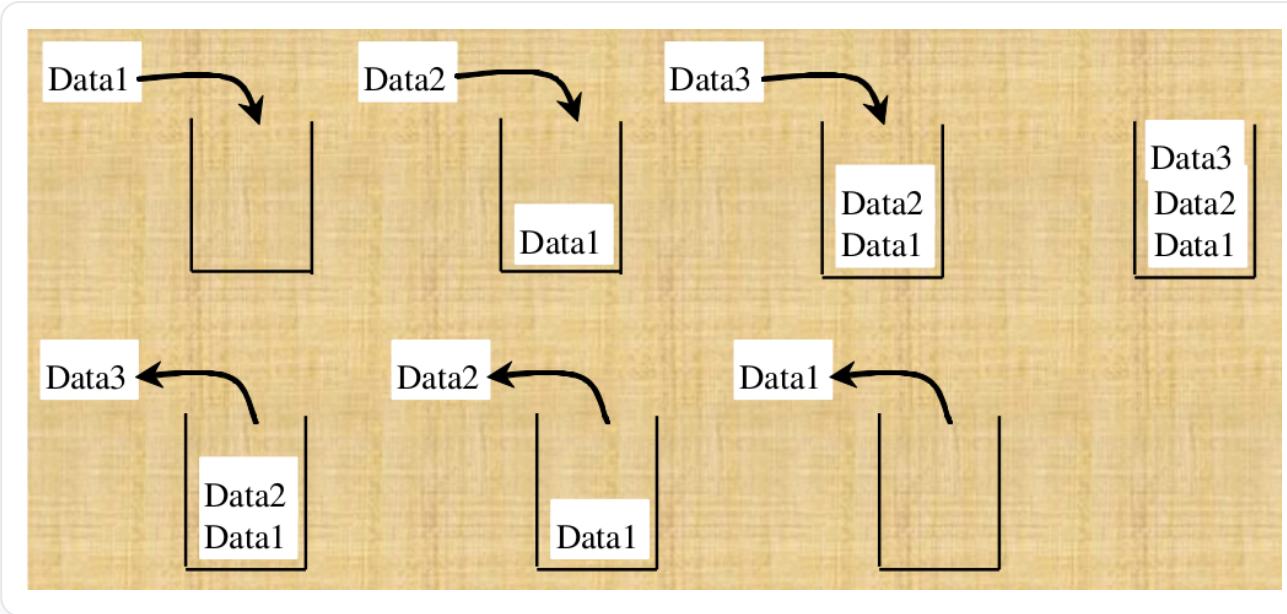
        System.out.println("Number of students in course1: "
            + course1.getNumberOfStudents());
        String[] students = course1.getStudents();
        for (int i = 0; i < course1.getNumberOfStudents(); i++)
            System.out.print(students[i] + ", ");

        System.out.println();
        System.out.print("Number of students in course2: "
            + course2.getNumberOfStudents());
    }
}
```

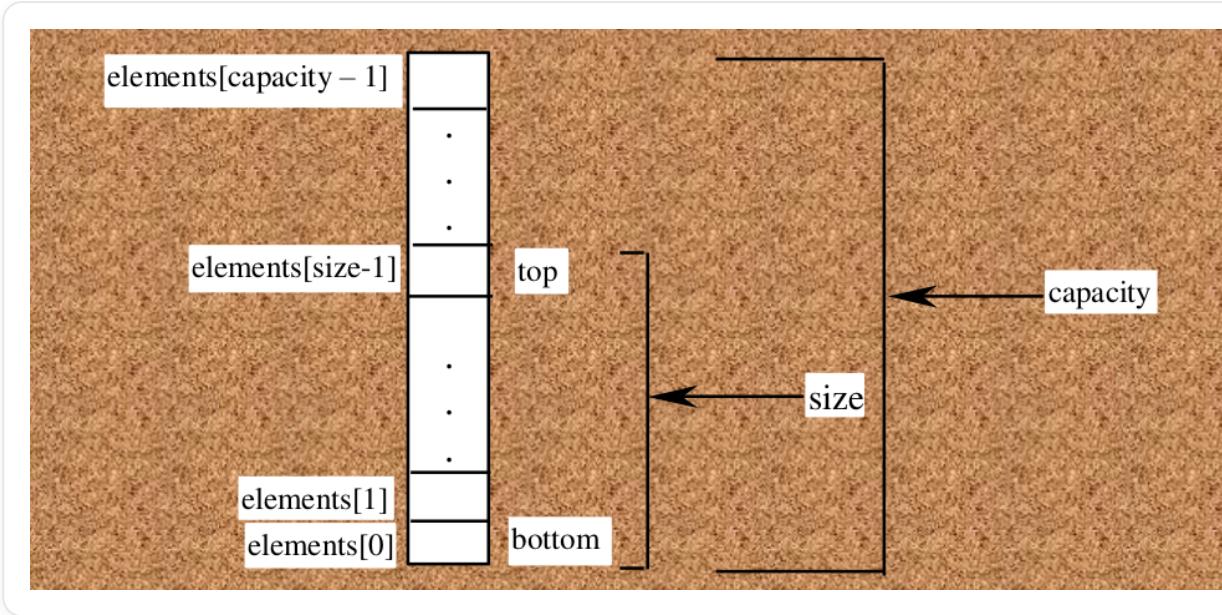
طراحی کلاس StackOfIntegers

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:



پیاده‌سازی:



```
In [ ]: 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;
        size++;
    }

    public int pop() {
        if (size == 0) {
            throw new IllegalStateException("Stack is empty");
        }
        int value = elements[size - 1];
        elements[size - 1] = 0;
        size--;
        return value;
    }

    public int peek() {
        if (size == 0) {
            throw new IllegalStateException("Stack is empty");
        }
        return elements[size - 1];
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public int getSize() {
        return size;
    }
}
```

```

        }
        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.

- استفاده می‌کنیم؟ **private** معمولاً از (fields) چرا برای تعریف ویژگی‌ها
- می‌تونیم امنیت داده‌ها رو حفظ کنیم؟ (Encapsulation) چطور با استفاده از کپسوله‌سازی

پایان

در صورت هرگونه سوال یا پیشنهاد میتوانید با من در

(:) ارتباط باشید

foroutanazanin@gmail.com telegram:

@naforutan