دانشکده مهندسی و علوم کامپیوتر

برنامه نویسی پیشرفته

وحیدی اصل

آشنایی با شیئ گرایی –بخش دوم

انگیزه

در جلسه قبل با برخی مزایای برنامه نویسی شیئ گرا آشنا شدیم.

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

اشیا و کلاسهای تغییرناپذیر (immutable)

- اگر محتوای یک شیئ پس از ایجاد آن نتواند تغییر کند، به آن شیئ تغییرناپذیر گفته می شود و کلاس مربوطه کلاس تغییرناپذیر نامیده می شود.
- اگر شما متد set در کلاس Circle را در اسلاید بعدی حذف کنید، کلاس تغییرناپذیر خواهد شد؛ چون radious سطح دسترسی private دارد و بدون متد set قابل دستکاری نمی باشد.

کلاسی که همه فیلدهای آن private هستند، بدون وجود تغییردهنده ها (mutators) لزوماً تغییرناپذیر نمی باشد. برای مثال کلاس Student که همه فیلدهای آن priavate است و متد تغییردهنده ندارد، همچنان تغییرپذیر می باشد.

مثالی از کلاس تغییرپذیر

```
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;
  }
}
```

مثالی از کلاس تغییرناپذیر

```
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 */
  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

```
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 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 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 داده ای آن باید private تعریف شوند و هیچ متد تغییردهنده یا متددسترسی وجود نداشته باشد که ارجاعی به یک فیلد داده ای قابل تغییر را برگردانند.

حوزه متغیرها

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

سطوح دسترسی در جاوا-بررسی دقیق تر

- تغییردهنده های دسترسی (access modifiers) در جاوا دسترس پذیری (حوزه) یک فیلد داده ای، متد، سازنده یا کلاس را مشخص می کند.
- در جاوا چهار نوع تغییردهنده دسترسی وجود دارد:

- private
- default
- protected
- public

تغییردهنده دسترسی private

- اگر سازنده یک کلاس را private تعریف کنیم، نمی توانیم نمونه ای 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 تغییردهنده ای استفاده نکنید، با آن فیلد یا متد، به طور پیش فرض default برخورد می شود. این تغییردهنده سبب می شود فیلد یا متد مربوطه تنها از درون پکیجش قابل دسترسی باشد.
- در مثال زیر دو پکیج pack و mypack داریم:

```
//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 می شود فیلد مربوطه درون پکیج قابل دسترسی باشد یا توسط وراثت در خارج پکیج دسترس پذیر شود.
- این تغییردهنده می تواند برروی فیلد داده ای، متد یا سازنده اعمال شود و برروی کلاس اعمال نمی شود.
- 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 msg در این پکیج به صورت protected تعریف شده، در نتیجه در خارج پکیج فقط با وراثت قابل دسترسی می باشد:

```
//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 تغییردهنده ها 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
```

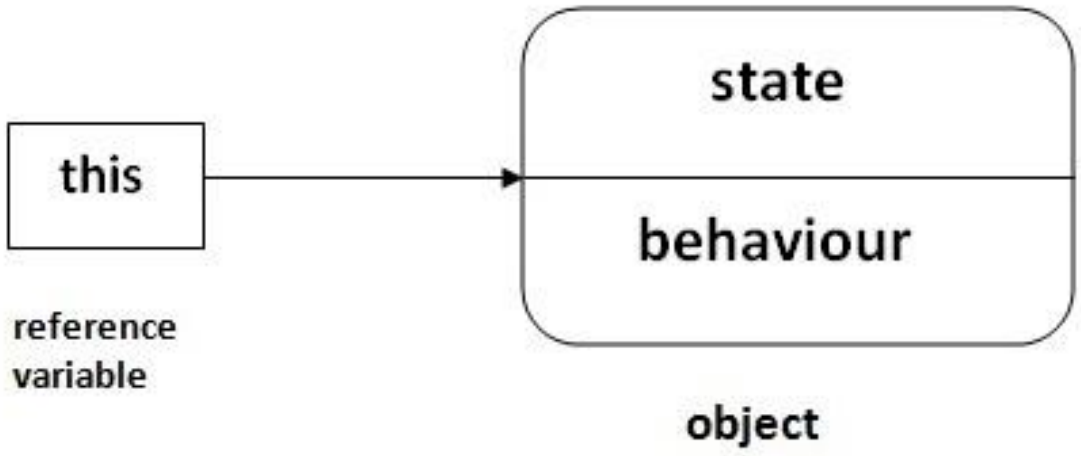
جدول سطوح دسترسی در جاوا

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 شیئ اشاره می کند.
- کاربرد مهم آن ارجاع به متغیرنمونه کلاس فعلی (جاری) است.
- This() برای فراخوانی سازنده کلاس فعلی استفاده می شود.
- برای فراخوانی متد کلاس فعلی به طور ضمنی استفاده می شود.
- می تواند به عنوان آرگومان در فراخوانی یک متد استفاده شود.
- می تواند به عنوان آرگومان در فراخوانی سازنده ارسال شود.
  - به یک سازنده امکان می دهد، سازنده دیگری از همان کلاس را فراخوانی نماید.
- برای برگرداندن نمونه کلاس فعلی استفاده می شود.

کلمه کلیدی this به نمونه فعلی کلاس اشاره می کند.



ارجاع به متغیرنمونه کلاس فعلی

هرگاه میان متغیرنمونه و پارامتر ارسالی، ابهام وجود داشته باشد، this برای رفع ابهام به کار می رود.

اگر در مثال زیر از this استفاده نکنیم:

```
class Student10{
    int id;
    String name;

    student(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 متغیرنمونه کلاس فعلی به عنوان فیلد پنهان

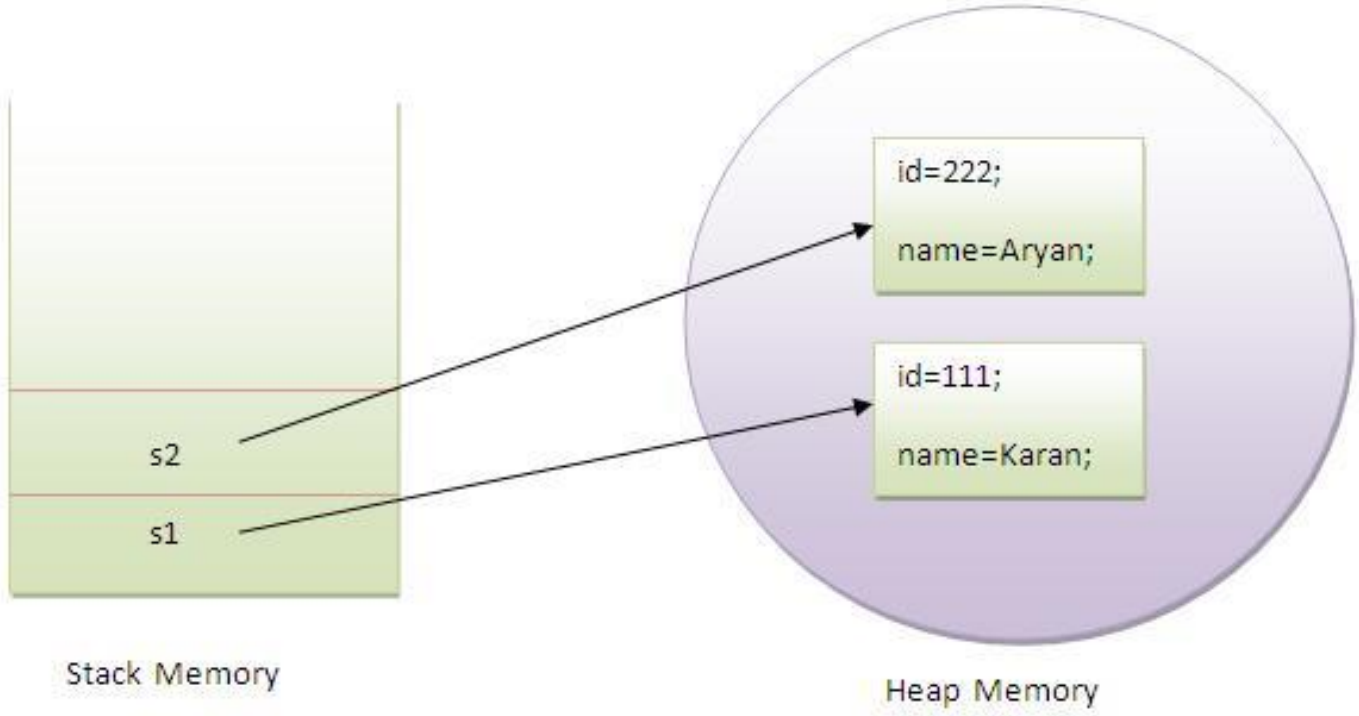
در مثال قبلی، پارامتر متد (آرگومانهای فرمال) و متغیرهای نمونه یکی هستند و به همین دلیل برای تمایز آنها از کلمه this استفاده می شود.

```
//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();
}
}
```

Test it Now

```
Output111 Karan
          222 Aryan
```

کلمه کلیدی this به فیلدهای داده ای اشیای کلاس جاری اشاره می کند



اگر نام متغیرهای نمونه و پارامترهای متدکلاس متفاوت باشند

در اینصورت 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();
}
}
```

استفاده از this برای فراخوانی سازنده کلاس فعلی

- فراخوانی سازنده this() می تواند برای فراخوانی سازنده کلاس فعلی (زنجیره سازنده ها) استفاده شود.

```
//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 invoked 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();
   }
}
```

Test it Now

چه موقع از فراخوانی سازنده this() استفاده کنیم؟

- فراخوانی سازنده this() در مواقع استفاده مجدد از یک سازنده در سازنده دیگر استفاده می شود.
- این کلمه زنجیره ای را میان سازنده ها برقرار می کند.

```
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();
   }
}
```

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();
   }
}
```

- شما می توانید متدی 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 متد اضافه می کند. برای مثال:

```
class A{

void m(){}

void n(){
m();
}

public static void main(String args[]){

new A().n();

}}
```

مثالی 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();//complier will add this to invoke n() method as this.n()
  }
  public static void main(String args[]){
  S s1 = new S();
  s1.p();
  }
}
```

Test it Now

```
Output:method is invoked
```

آیا this دقیقا همان ریموت کنترل شیئ است؟

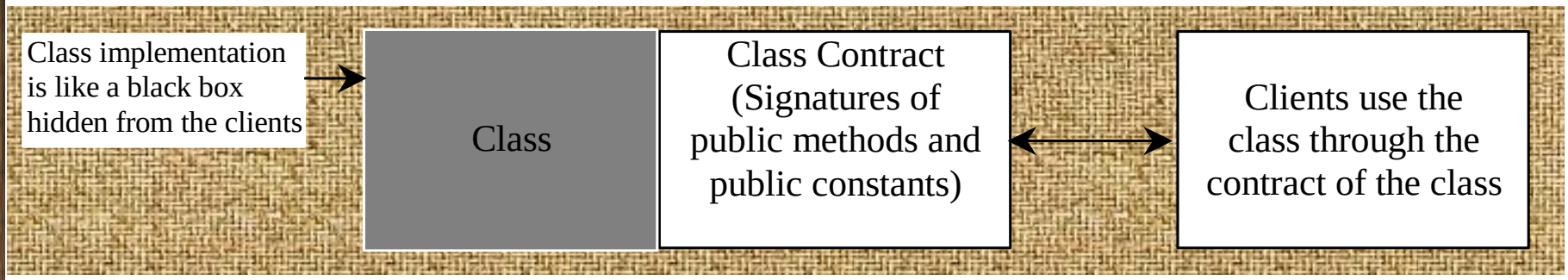
- برای اثبات اینکه this به متغیرنمونه کلاس فعلی اشاره می کند، برنامه زیر را ملاحظه کنید:

```
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();
}
}
```

Test it Now

```
Output:A5@22b3ea59
       A5@22b3ea59
```

- انتزاع کلاس به معنای تفکیک پیاده سازی کلاس از استفاده آن کلاس است.
- ایجادکننده کلاس توصیف (توضیحاتی) از کلاس را ارایه می کند و به کاربر می گوید چگونه از کلاس استفاده کند.
- کاربر کلاس نیاز نیست بداند کلاس چگونه نوشته شده (پیاده سازی شده است). جزئیات پیاده سازی بسته بندی (کپسوله بندی) شده از دید کاربر پنهان است.



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.

طراحی کلاس وام

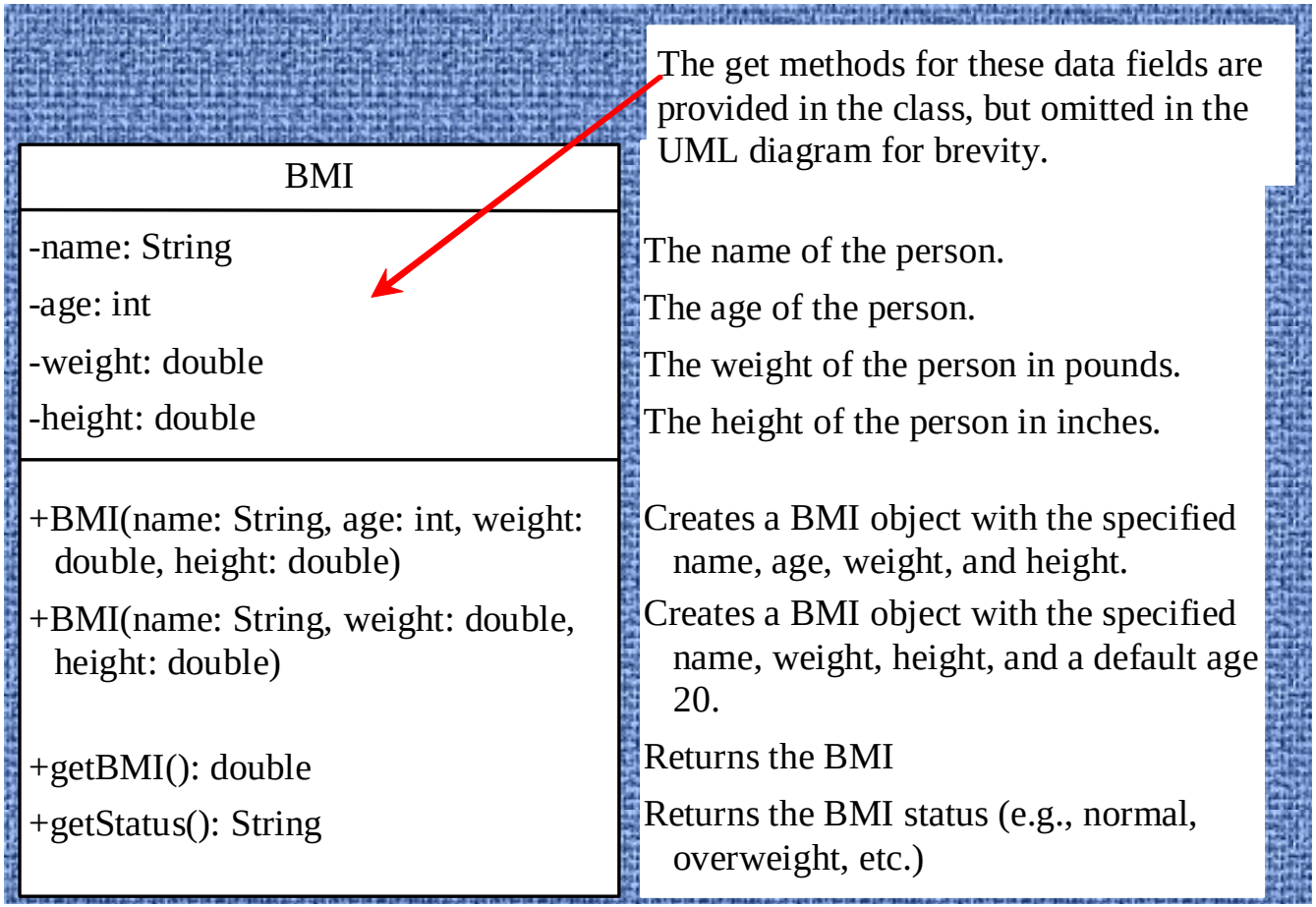
```
public class Loan {
  private double annualInterestRate;
  private int numberOfYears;
  private double loanAmount;
  private java.util.Date loanDate;

  /** 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 newloanAmount */
  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 وام

```
import java.util.Scanner;

public class TestLoanClass {
  /** Main method */
  public static void main(String[] args) {
    // 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 numberOfYears = 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());
  }
}
```

BMI (Body Mass Index) کلاس



BMI کلاس

```
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;
  }
}
```

ClassUSeBMI کلاس

```
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());
  }
}
```

مثال : 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.

Course کلاس

```
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;
  }
}
```

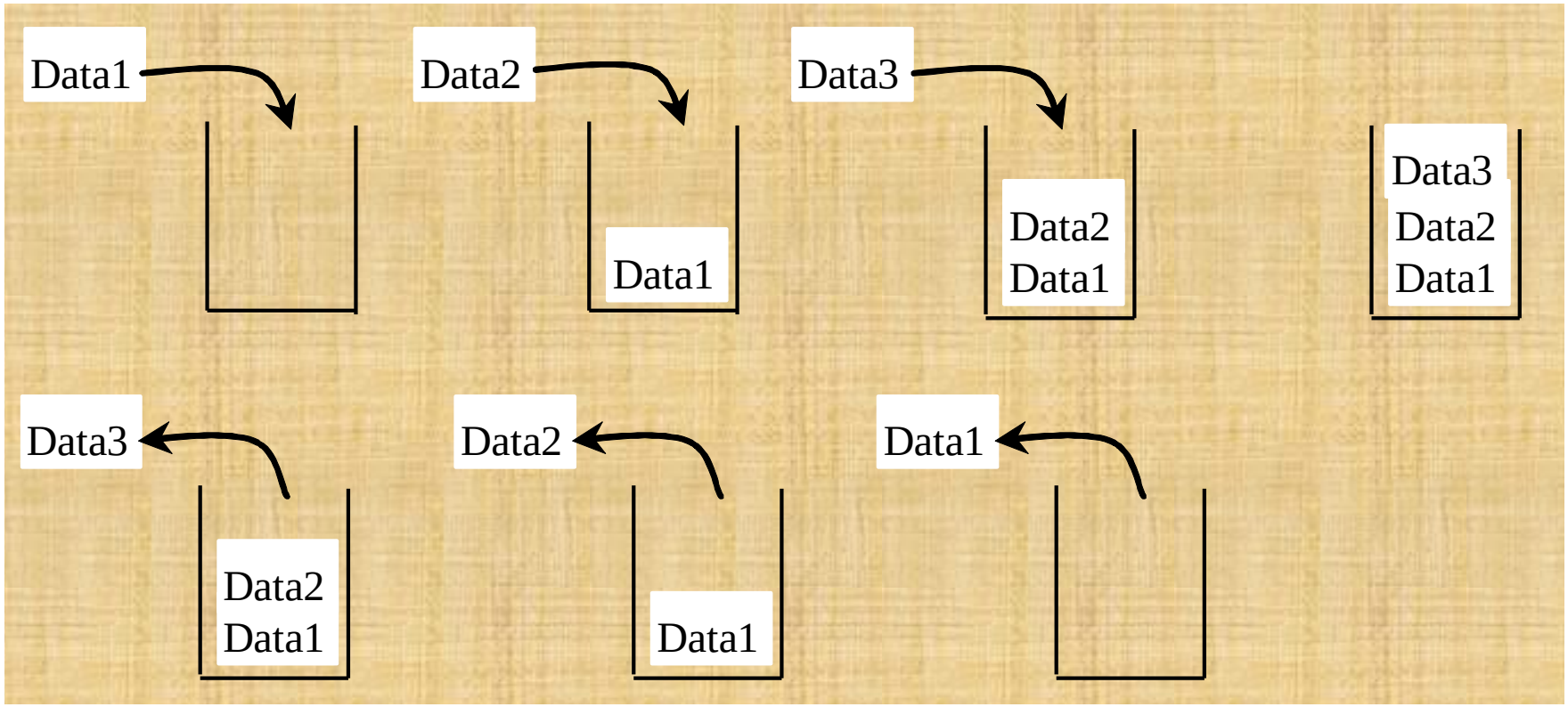
TestCourse کلاس

```
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());
  }
}
```

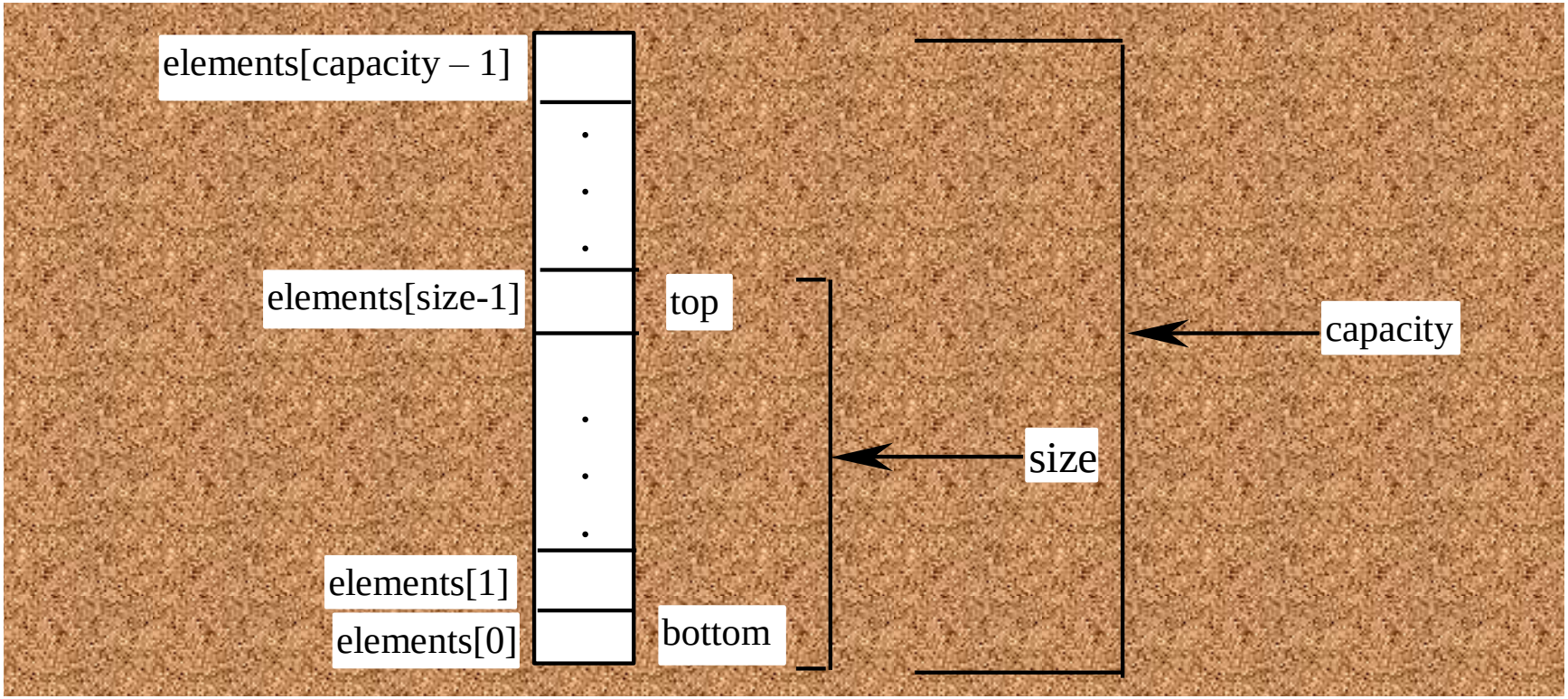
مثال: کلاس StackOfIntegers

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.

StackOfIntegers طراحی کلاس



StackOfIntegers پیاده سازی کلاس



پشته اعداد صحیح

```
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;
  }
}
```