

Лекция 12.1

Наследственост в ООП

Част II

Основни теми

- Реализация на многократно използване на софтуер посредством прилагане на концепцията за наследственост.
- Дефиниция и приложения на базов клас и производни класове.
- Използване на `extends` за създаване на клас, който онаследява данните и поведението на друг клас.
- Използване на модификатор `protected` за позволяване производен клас да получи достъп до членовете на базов клас.
- **Получаване на достъп до членовете на базовия клас чрез ключовата дума `super`.**
- **Поведение на конструкторите при класове в йерархия от наследственост.**
- **Методи на клас `Object` и онаследяването им във всеки клас.**

- 9.5 Конструктори на производни класове
- 9.6 Software Engineering с наследственост
- 9.7 Класът Object
- 9.8 GUI и Graphics пример: Извеждане на текст и изображения с етикети

Задачи

Литература:

Java How to Program, Sixth Edition, глава 9

9.5 Конструктори на производни класове

- Инициализиране на обекти от производни класове
 - Каскадно (верижно) извикване на конструктори
 - Всеки конструктор на производен клас извиква конструктора на своя директен базов клас (**явно или неявно**)
 - Дървовидна структура на наследственост
 - Конструкторът на класът на най- високото ниво от йерархията извиква конструкторът на клас `Object` (**защо?**)
 - Конструкторът на текущия производен клас се изпълнява последен
 - Пример: йерархия на наследственост
CommissionEmployee3 - BasePlusCommissionEmployee4
 - Конструкторът на `CommissionEmployee3` се изпълнява след като приключи изпълнението на конструктора на клас `Object`
 - Конструкторът на `BasePlusCommissionEmployee4` се изпълнява след като приключи изпълнението последователно на конструкторите на клас `Object` и на клас `CommissionEmployee3`

Software Engineering факт 9.8

При създаване на обект от произведен клас, конструкторът на този клас изпълнява като първа команда конструкторът на директния базов клас (**явно**, чрез ключовата дума ***super***, или **неявно** като компилаторът вмъква необходимите за целта команди).

При това, **конструкторът на базовият клас** трябва да инициализира всичките данни на базовия клас, *онаследени в производния клас*, а **конструкторът на производния клас** инициализира останалите данни специфични за производния клас (*всички останали данни, различни от онаследените*)

Software Engineering факти 9.8

При **липса на извикване на конструктор на базов клас** като първа команда в конструктора на базовия клас и **липса на конструктор в базовия клас** се извършва **инициализиране на онаследените данни по подразбиране** (т.е. 0 *примитивни числови данни*, false за *булеви данни*, null за *всички данни от референтен тип*).

Outline

CommissionEmployee4
.java

(1 от 4)

Редове 23-24

```
1 // Fig. 9.15: CommissionEmployee4.java
2 // CommissionEmployee4 class represents a commission employee.
3
4 public class CommissionEmployee4
5 {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9     private double grossSales; // gross weekly sales
10    private double commissionRate; // commission percentage
11
12    // five-argument constructor
13    public CommissionEmployee4( String first, String last, String ssn,
14        double sales, double rate )
15    {
16        // implicit call to Object constructor occurs here
17        firstName = first;
18        lastName = last;
19        socialSecurityNumber = ssn;
20        setGrossSales( sales ); // validate and store gross sales
21        setCommissionRate( rate ); // validate and store commission rate
22
23        System.out.printf(
24            "\nCommissionEmployee4 constructor:\n%s\n", this );
25    } // end five-argument CommissionEmployee4 constructor
26
```

Неявно извикване на
конструктор по подразбиране
на базовия клас (class Object).

Извеждане на съобщение за
проследяване на
последователността в
извикването на
конструкторите.



Outline

CommissionEmployee4
.java

(2 of 4)

```
27 // set first name
28 public void setFirstName( String first )
29 {
30     firstName = first;
31 } // end method setFirstName
32
33 // return first name
34 public String getFirstName()
35 {
36     return firstName;
37 } // end method getFirstName
38
39 // set last name
40 public void setLastName( String last )
41 {
42     lastName = last;
43 } // end method setLastName
44
45 // return last name
46 public String getLastName()
47 {
48     return lastName;
49 } // end method getLastName
50
51 // set social security number
52 public void setSocialSecurityNumber( String ssn )
53 {
54     socialSecurityNumber = ssn; // should validate
55 } // end method setSocialSecurityNumber
56
```



Outline

CommissionEmployee 4.java

(3 от 4)

```
57 // return social security number
58 public String getSocialSecurityNumber()
59 {
60     return socialSecurityNumber;
61 } // end method getSocialSecurityNumber
62
63 // set gross sales amount
64 public void setGrossSales( double sales )
65 {
66     grossSales = ( sales < 0.0 ) ? 0.0 : sales;
67 } // end method setGrossSales
68
69 // return gross sales amount
70 public double getGrossSales()
71 {
72     return grossSales;
73 } // end method getGrossSales
74
75 // set commission rate
76 public void setCommissionRate( double rate )
77 {
78     commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
79 } // end method setCommissionRate
80
```



Outline

CommissionEmployee4
.java

(4 of 4)

```
81 // return commission rate
82 public double getCommissionRate()
83 {
84     return commissionRate;
85 } // end method getCommissionRate
86
87 // calculate earnings
88 public double earnings()
89 {
90     return getCommissionRate() * getGrossSales();
91 } // end method earnings
92
93 // return String representation of CommissionEmployee4 object
94 public String toString()
95 {
96     return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
97         "commission employee", getFirstName(), getLastName(),
98         "social security number", getSocialSecurityNumber(),
99         "gross sales", getGrossSales(),
100        "commission rate", getCommissionRate() );
101 } // end method toString
102 } // end class CommissionEmployee4
```



Outline

Онаследява class
CommissionEmployee4

BasePlusCommission
Employee5.java

(1 от 2)

Редове 15-16

```

1 // Fig. 9.16: BasePlusCommissionEmployee5.java
2 // BasePlusCommissionEmployee5 class declaration.
3
4 public class BasePlusCommissionEmployee5 extends CommissionEmployee4
5 {
6     private double baseSalary; // base salary per week
7
8     // six-argument constructor
9     public BasePlusCommissionEmployee5( String first, String last,
10         String ssn, double sales, double rate, double salary )
11     {
12         super( first, last, ssn, sales, rate );
13         setBaseSalary( salary ); // validate and store base salary
14
15         System.out.printf(
16             "\nBasePlusCommissionEmployee5 constructor:\n%s\n", this );
17     } // end six-argument BasePlusCommissionEmployee5 constructor
18
19     // set base salary
20     public void setBaseSalary( double salary )
21     {
22         baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
23     } // end method setBaseSalary
24

```

Извеждане на съобщение за
проследяване на
последователността в
извикването на
конструкторите.



Outline

BasePlusCommission
Employee5.java

(2 от 2)

```
25 // return base salary
26 public double getBaseSalary()
27 {
28     return baseSalary;
29 } // end method getBaseSalary
30
31 // calculate earnings
32 public double earnings()
33 {
34     return getBaseSalary() + super.earnings();
35 } // end method earnings
36
37 // return String representation of BasePlusCommissionEmployee5
38 public String toString()
39 {
40     return String.format( "%s %s\n%s: %.2f", "base-salaried",
41                             super.toString(), "base salary", getBaseSalary() );
42 } // end method toString
43 } // end class BasePlusCommissionEmployee5
```



Outline

```

1 // Fig. 9.17: ConstructorTest.java
2 // Display order in which superclass and subclass constructors are called.
3
4 public class ConstructorTest
5 {
6     public static void main( String args[] )
7     {
8         CommissionEmployee4 employee1 = new CommissionEmployee4(
9             "Bob", "Lewis", "333-33-3333", 5000, .04 );
10
11         System.out.println();
12         BasePlusCommissionEmployee5 employee2 =
13             new BasePlusCommissionEmployee5(
14                 "Lisa", "Jones", "555-55-5555", 2000, .06, 800 );
15
16         System.out.println();
17         BasePlusCommissionEmployee5 employee3 =
18             new BasePlusCommissionEmployee5(
19                 "Mark", "Sands", "888-88-8888", 8000, .15, 2000 );
20     } // end main
21 } // end class ConstructorTest

```

Създава обект от клас
CommissionEmployee4

Създава два обекта от клас
BasePlusCommissionEmployee5
за илюстриране на
последователността в извикването на
конструкторите.

ConstructorTest

.java

(1 от 2)

Редове 8-9

Редове 12-19



Outline

ConstructorTest

.java

(2 от 2)

Конструкторът на производния клас
BasePlusCommissionEmployee5
се изпълнява след конструктора на
базовия клас
CommissionEmployee4.

CommissionEmployee4 constructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04

CommissionEmployee4 constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 0.00

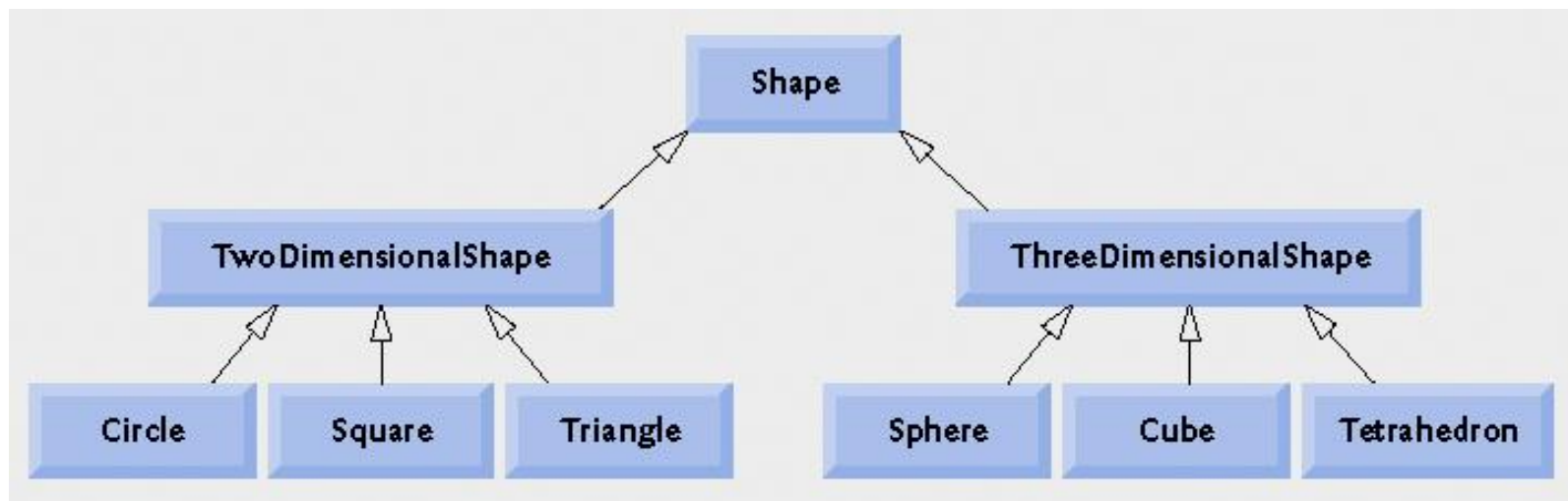
BasePlusCommissionEmployee5 constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: 2000.00
commission rate: 0.06
base salary: 800.00

CommissionEmployee4 constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 0.00

BasePlusCommissionEmployee5 constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: 8000.00
commission rate: 0.15
base salary: 2000.00



9.5 Правила за писане на конструктори на класове в йерархия на наследственост



9.5 Правила за писане на конструктори на класове в йерархия на наследственост

1. Пишем **базовия клас** на йерархията от наследственост по правилата за моделиране на клас, дадени в **лекция 11.1** в следната последователност
 - A. *private* клас данни
 - B. SET и GET методи за всички клас данни
 - C. Конструктор за общо ползване (извиква set методите за данните)
 - D. Конструктор по подразбиране (извиква конструктора за общо ползване)
 - E. Конструктор за копиране (извиква конструктора за общо ползване)
 - F. **Всички останали клас методи**
 - G. *String toString()* метод

9.5 Правила за писане базов клас-деклариране на данните

```
// Fig. 9.15a: CommissionEmployee4.java
// CommissionEmployee4 class represents a commission employee.

public class CommissionEmployee4
{
    private String firstName;
    private String lastName;
    private String socialSecurityNumber;
    private double grossSales; // gross weekly sales
    private double commissionRate; // commission percentage
```

9.5 Правила за писане базов клас-SET и GET методи

```
// set first name
public void setFirstName( String first )
{
    firstName = first;
} // end method setFirstName

// return first name
public String getFirstName()
{
    return firstName;
} // end method getFirstName

// set last name
public void setLastName( String last )
{
    lastName = last;
} // end method setLastName

// return last name
public String getLastName()
{
    return lastName;
} // end method getLastName
```

9.5 Правила за писане базов клас-SET и GET методи ...

```
// set social security number
public void setSocialSecurityNumber( String ssn )
{
    socialSecurityNumber = ssn; // should validate
} // end method setSocialSecurityNumber

// return social security number
public String getSocialSecurityNumber()
{
    return socialSecurityNumber;
} // end method getSocialSecurityNumber

// set gross sales amount
public void setGrossSales( double sales )
{
    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end method setGrossSales

// return gross sales amount
public double getGrossSales()
{
    return grossSales;
} // end method getGrossSales
```

9.5 Правила за писане базов клас-SET и GET методи ...

```
// set commission rate
public void setCommissionRate( double rate )
{
    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end method setCommissionRate

// return commission rate
public double getCommissionRate()
{
    return commissionRate;
} // end method getCommissionRate
```

9.5 Правила за писане базов клас-конструктор за общо ползване

```
private String firstName;
private String lastName;
private String socialSecurityNumber;
private double grossSales;          // gross weekly sales
private double commissionRate;      // commission percentage
// five-argument constructor
public CommissionEmployee4( String first, String last, String ssn,
                           double sales, double rate )
{
    // implicit call to Object constructor occurs here
    firstName = first;
    lastName = last;
    socialSecurityNumber = ssn;
    setGrossSales( sales ); // validate and store gross sales
    setCommissionRate( rate ); // validate and store commission rate

    System.out.printf(
        "\nCommissionEmployee4 constructor:\n%s\n", this );
} // end five-argument CommissionEmployee4 constructor
```

9.5 Правила за писане базов клас-конструктори по подразбиране и за копиране

```
// default constructor
public CommissionEmployee4( )
{
    this("", "", "", 0.0, 0.0);
} // end five-argument CommissionEmployee4 constructor

// copy constructor
public CommissionEmployee4(CommissionEmployee4 c )
{
    this(c.firstName, c.lastName, c.socialSecurityNumber,
        c.grossSales, c.commissionRate);
} // end five-argument CommissionEmployee4 constructor
```

9.5 Правила за писане базов клас-други методи на класа и *toString()* метода

```
// calculate earnings
public double earnings()
{
    return getCommissionRate() * getGrossSales();
} // end method earnings

// return String representation of CommissionEmployee4 object
public String toString()
{
    return String.format( "%s: %s %s\n%s: %s\n%s: %.2f\n%s: %.2f",
        "commission employee", getFirstName(), getLastName(),
        "social security number", getSocialSecurityNumber(),
        "gross sales", getGrossSales(),
        "commission rate", getCommissionRate() );
} // end method toString
```

9.5 Правила за писане на конструктори на класове в йерархия на наследственост

2. Пишем всеки от производните класове на йерархията от наследственост по следните правила в следната последователност
 - A. Декларира всички **private** клас данни, които са различни от онаследените
 - B. SET и GET методи за всички клас данни, които са различни от онаследените
 - C. Конструктор за общо ползване
 - a. Извиква конструкторът за общо ползване на директния базов клас и инициализира ВСИЧКИ онаследени данни
 - b. извиква set методите за данните, които са различни от онаследените
 - D. Конструктор по подразбиране (извиква **конструктора за общо ползване на текущия клас**, задава **стойности по подразбиране за всички** данни – онаследени и тези, дефинирани в текущия клас)
 - E. Конструктор за копиране (извиква **конструктора за общо ползване на текущия клас** , използва **GET методи за онаследени клас данни**)
 - F. **Всички останали клас методи**
 - G. *String toString()* метод

9.5 Правила за писане произведен клас- - деклариране на новите данни

```
// Fig. 9.16a: BasePlusCommissionEmployee5.java  
// Modified BasePlusCommissionEmployee5 class declaration.
```

```
public class BasePlusCommissionEmployee5 extends CommissionEmployee4  
{  
    // Тук не се декларира отново данните, които са онаследени!  
    private double baseSalary; // base salary per week
```

9.5 Правила за писане производен клас SET и GET методи само за новите данни

```
// set base salary
public void setBaseSalary( double salary )
{
    baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end method setBaseSalary

// return base salary
public double getBaseSalary()
{
    return baseSalary;
} // end method getBaseSalary
```

9.5 Правила за писане произведен клас

Конструктор за общо ползване

```
// six-argument constructor- инициализира ВСИЧКИ данни
public BasePlusCommissionEmployee5( String first, String last,
    String ssn, double sales, double rate, double salary )
{
    super( first, last, ssn, sales, rate ); // инициализира онаследените
    // следва инициализация на всички данни, които не са онаследени
    setBaseSalary( salary ); // validate and store base salary

    System.out.printf(
        "\nBasePlusCommissionEmployee5 constructor:\n%s\n", this );
} // end six-argument BasePlusCommissionEmployee5 constructor
```

9.5 Правила за писане произведен клас

Конструктори по подразбиране и за копиране

```
// default constructor
public BasePlusCommissionEmployee5( )
{
    this("", "", "", 0.0, 0.0, 0.0);
} // end six-argument BasePlusCommissionEmployee5 constructor

// default constructor
public BasePlusCommissionEmployee5( BasePlusCommissionEmployee5 b)
{
    this( b.getFirstName(), b.getLastName(),
          b.getSocialSecurityNumber(),
          b.getGrossSales(), b.getCommissionRate(), b.baseSalary);
} // end six-argument BasePlusCommissionEmployee5 constructor
```

9.5 Правила за писане произведен клас други методи на класа и *toString()* метода

```
// calculate earnings
public double earnings()
{
    return getBaseSalary() + super.earnings();
} // end method earnings

// return String representation of BasePlusCommissionEmployee5
public String toString()
{
    return String.format( "%s %s\n%s: %.2f", "base-salaried",
        super.toString(), "base salary", getBaseSalary() );
} // end method toString
```

9.6 Използване на наследственост

- **Признаци за необходимост от наследственост**
 - Едни и същи данни се срещат в два или повече класа
 - Едни и същи методи, които евентуално оперират една и съща група данни се срещат в два или повече класа
- **Приложение на наследственост**
 - Обособяване на базов клас с общите данни и методи
 - Дефиниране на йерархия от наследственост
 - Не се изисква достъп до данните на базовия клас
 - Независима поддръжка и промяна на кода

9.7 клас Object

- Методите на клас Object
 - clone
 - equals
 - finalize
 - getClass
 - hashCode
 - notify, notifyAll, wait
 - toString

Method	Description
<code>clone</code>	<p>This protected method, which takes no arguments and returns an <code>Object</code> reference, makes a copy of the object on which it is called. When cloning is required for objects of a class, the class should override method <code>clone</code> as a public method and should implement interface <code>Cloneable</code> (package <code>java.lang</code>). The default implementation of this method performs a so-called shallow copy—instance variable values in one object are copied into another object of the same type. For reference types, only the references are copied. A typical overridden <code>clone</code> method's implementation would perform a deep copy that creates a new object for each reference type instance variable. There are many subtleties to overriding method <code>clone</code>. You can learn more about cloning in the following article:</p> <p>java.sun.com/developer/JDCTechTips/2001/tt0306.html</p>

Fig. 9.18 | Методите на клас `Object` се онаследяват явно или неявно от всички класове.
(1 от 4)

Method	Description
<code>equals</code>	<p>This method compares two objects for equality and returns <code>true</code> if they are equal and <code>false</code> otherwise. The method takes any <code>Object</code> as an argument. When objects of a particular class must be compared for equality, the class should override method <code>equals</code> to compare the contents of the two objects. The method's implementation should meet the following requirements:</p> <ul style="list-style-type: none"> • It should return <code>false</code> if the argument is <code>null</code>. • It should return <code>true</code> if an object is compared to itself, as in <code>object1.equals(object1)</code>. • It should return <code>true</code> only if both <code>object1.equals(object2)</code> and <code>object2.equals(object1)</code> would return <code>true</code>. • For three objects, if <code>object1.equals(object2)</code> returns <code>true</code> and <code>object2.equals(object3)</code> returns <code>true</code>, then <code>object1.equals(object3)</code> should also return <code>true</code>. • If <code>equals</code> is called multiple times with the two objects and the objects do not change, the method should consistently return <code>true</code> if the objects are equal and <code>false</code> otherwise. <p>A class that overrides <code>equals</code> should also override <code>hashCode</code> to ensure that equal objects have identical hashcodes. The default <code>equals</code> implementation uses operator <code>==</code> to determine whether two references <i>refer to the same object</i> in memory. Section 29.3.3 demonstrates class <code>String</code>'s <code>equals</code> method and differentiates between comparing <code>String</code> objects with <code>==</code> and with <code>equals</code>.</p>

Fig. 9.18 | Методите на клас `Object` се онаследяват явно или неявно от всички класове.
(2 от 4)

Method	Description
<code>finalize</code>	This protected method (introduced in Section 8.10 and Section 8.11) is called by the garbage collector to perform termination housekeeping on an object just before the garbage collector reclaims the object's memory. It is not guaranteed that the garbage collector will reclaim an object, so it cannot be guaranteed that the object's <code>finalize</code> method will execute. The method must specify an empty parameter list and must return <code>void</code> . The default implementation of this method serves as a placeholder that does nothing.
<code>getClass</code>	Every object in Java knows its own type at execution time. Method <code>getClass</code> (used in Section 10.5 and Section 21.3) returns an object of class <code>Class</code> (package <code>java.lang</code>) that contains information about the object's type, such as its class name (returned by <code>Class</code> method <code>getName</code>). You can learn more about class <code>Class</code> in the online API documentation at java.sun.com/j2se/5.0/docs/api/java/lang/Class.html .

Fig. 9.18 | Методите на клас `Object` се онаследяват явно или неявно от всички класове. (3 от 4)

Method	Description
hashCode	A hashtable is a data structure (discussed in Section 19.10) that relates one object, called the key, to another object, called the value. When initially inserting a value into a hashtable, the key's hashCode method is called. The hashcode value returned is used by the hashtable to determine the location at which to insert the corresponding value. The key's hashcode is also used by the hashtable to locate the key's corresponding value.
notify, notifyAll, wait	Methods notify , notifyAll and the three overloaded versions of wait are related to multithreading, which is discussed in Chapter 23. In J2SE 5.0, the multithreading model has changed substantially, but these features continue to be supported.
toString	This method (introduced in Section 9.4.1) returns a String representation of an object. The default implementation of this method returns the package name and class name of the object's class followed by a hexadecimal representation of the value returned by the object's hashCode method.

Fig. 9.18 | Object methods that are inherited directly or indirectly by all classes. (Part 4 of 4)

9.8 GUI и Graphics пример: Извеждане на текст и картина с етикети

- **Етикети**
 - Служат за извеждане на информация и инструкции
 - **JLabel**
 - Изобразява в графичен формат текст
 - Изобразява картинка
 - Изобразява текст и картинка

Outline

LabelDemo.java

(1 от 2)

Ред 13

Ред 16

Ред 19

Ред 25

```
1 // Fig 9.19: LabelDemo.java
2 // Demonstrates the use of labels.
3 import java.awt.BorderLayout;
4 import javax.swing.ImageIcon;
5 import javax.swing.JLabel;
6 import javax.swing.JFrame;
7
8 public class LabelDemo
9 {
10     public static void main( String args[] )
11     {
12         // Create a label with plain text
13         JLabel northLabel = new JLabel( "North" );
14
15         // create an icon from an image so we can put it on a JLabel
16         ImageIcon labelIcon = new ImageIcon( "GUItip.gif" );
17
18         // create a label with an icon instead of text
19         JLabel centerLabel = new JLabel( labelIcon );
20
21         // create another label with an icon
22         JLabel southLabel = new JLabel( labelIcon );
23
24         // set the label to display text (as well as an icon)
25         southLabel.setText( "South" );
26     }
```

Създава JLabel и
извежда текста "North"

Конструктор на
ImageIcon задава пътя
да картинката

Декларира и инициализира
етикет centerLabel за
извеждане на labelIcon

Променя текста в
етикета southLabel



Outline

LabelDemo.java

(2 от 2)

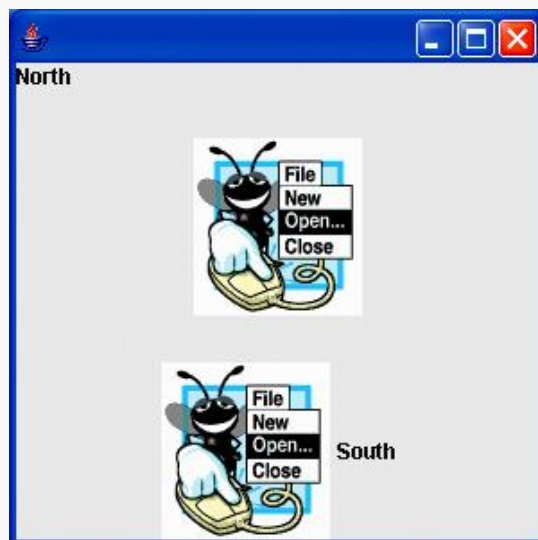
Редове 34-36

```

27 // create a frame to hold the labels
28 JFrame application = new JFrame();
29
30 application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
31
32 // add the labels to the frame; the second argument specifies
33 // where on the frame to add the label
34 application.add( northLabel, BorderLayout.NORTH );
35 application.add( centerLabel, BorderLayout.CENTER );
36 application.add( southLabel, BorderLayout.SOUTH );
37
38 application.setSize( 300, 300 ); // set the size of the frame
39 application.setVisible( true ); // show the frame
40 } // end main
41 } // end class LabelDemo

```

Добавя етикети към JFrame в разделите наречени north, center and south



Задачи

1. Напишете следните класове като **спазвате концепциите** за скриване на информация (*encapsulation, information hiding*) и изискване за многократно използване на код (*software reuse-избягване на дублиране на код!*)
 - Напишете *class Point*, чиито обекти имат две данни-координатите *x* и *y* на точката като цели числа.
 - Напишете пълен набор от конструктори за *class Point*, *set* и *get* методи за данните на класа, а също и съответен *toString()* метод.

Задачи

- Напишете *class Rectangle* като реализирате следната дефиниция за моделиране на класа- *Rectangle* е *Point* , която дефинира горния ляв ъгъл на *Rectangle* и има *Point* , която дефинира долния десен ъгъл на *Rectangle*.
- Напишете пълен набор от конструктори за *class Rectangle* , set и get методи за данните на класа, а също и съответен *toString()* метод.
- Напишете в метод а *double diagonal()* в *class Rectangle* , позволяващ да се сметне диагонала на текущия обект от *class Rectangle*.

Задачи

- **Напишете Java Console application за тестване на йерархията от наследственост *Point- Rectangle***
(създайте два обекта *Points*, създайте обект *Rectangle* посредством тези точки, изведете текущите данни на тези обекти, а също и дължината на диагонала на *създадения* обект *Rectangle*.)

Задачи

2. Напишете следните класове като **спазвате концепциите** за скриване на информация (*encapsulation, information hiding*) и изискване за многократно използване на код (*software reuse-избягване на дублиране на код!*)
- Напишете *class Page* (страница на книга), чиито обекти имат масив от низове *rows[]* (представя редовете на страницата).
 - Напишете пълен набор от конструктори за *class Page*, *set* и *get* методи за данните на класа, а също и съответен *toString()* метод.

Задачи

- Напишете *class Chapter* (глава на книга) като реализирате следната дефиниция за моделиране на класа-*class Chapter* е *class Page*, който има *title* (заглавие) който е низ от текст и има уникален пореден номер (на главата).
- Напишете пълен набор от конструктори за *class Chapter*, *set* и *get* методи за данните на класа, а също и съответен *toString()* метод.
- Напишете в метод а *double printNumbers()* в *class Chapter*, позволяващ изведат редовете *rows* в номериран формат.

Задачи

- Напишете Java Console application за тестване на йерархията от наследственост *Page- Chapter*

(създайте масив от редове *rows[]* , създайте обект от клас *Chapter* последством тези редове , изведете тези редове на обекта от клас *Chapter* с пореден номер посредством метода *printNumbers()* , променете някой от елементите на първоначално създадения масив *rows[]* и отново изведете редовете на обекта от клас *Chapter* с пореден номер- **ако сте работили правилно** трябва да има съвпадане с предишното извеждане на тези редове под номер.)