

Лекция 12.2

Полиморфизъм в ООП Част I

Основни теми

- Представа за полиморфизъм и приложения.
- Използване на предефинирани методи като средство за постигане на полиморфизъм.
- Различия между абстрактни и конкретни класове.
- Деклариране на абстрактни методи при създаване на абстрактни класове.
- Предимства при използване на полиморфизъм-разширяемост на класове и улеснена поддръжка на софтуер.
- Определяне на типа на обектите по време на изпълнение на програмата.
- Деклариране и използване на *интерфейс*-и.

- 10.1 Въведение
- 10.2 Примери за полиморфизъм
- 10.3 Демонстриране на полиморфично поведение
- 10.4 **Абстрактни** класове и методи
- 10.5 Примери: Приложение за изчисляване на заплати
 - 10.5.1 Създаване на **абстрактен** базов клас Employee
 - 10.5.2 Създаване на **конкретен** производен клас SalariedEmployee
 - 10.5.3 Създаване на **конкретен** производен клас HourlyEmployee
 - 10.5.4 Създаване на **конкретен** производен клас CommissionEmployee
 - 10.5.5 Създаване на **индиректен (косвен) конкретен** производен клас BasePlusCommissionEmployee
 - 10.5.6 Демонстриране на **полиморфично поведение**, оператор instanceof и преобразуване “**надолу**” (downcasting)
 - 10.5.7 Обобщение на **позволените присвоявания** между променливи рефериращи базов и производен клас

Задачи

Литература:

Java How to Program, Sixth Edition, глава 10

10.6 final методи и класове

10.7 Примери: Дефиниране и използване на интерфейс- и

10.7.1 Моделиране на наследствена йерархия Payable за при решаване на задачата за пресмятане на заплати

10.7.2 Деклариране на interface Payable

10.7.3 Дефиниране на class Invoice

10.7.4 Редактиране на class Employee позволяващо да се приложи interface Payable

10.7.5 Редактиране на class SalariedEmployee позволяващо да се приложи interface Payable

10.7.6 Използване на interface Payable за полиморфична обработка на Invoice и Employee обекти

10.7.7 Деклариране на константи посредством interface

10.7.8 Някои често използвани интерфейси в Java API

Задачи

Литература:

Java How to Program, Sixth Edition, глава 10

10.1 Въведение

- **Полиморфизъм**

- **Позволява “програмиране на общото поведение” вместо “програмиране на частното, специфично поведение”**
- **Извикването на един и същи метод може да доведе до реализиране на “много форми” на изходния резултат**

- **интерфейси**

- **Прилага се към класове за реализиране на общо поведение с други (дори доста различни) класове**
- **Позволява да се осъществи явно онаследяване от повече от един субекта**

10.2 Примери за полиморфизъм

- **Примери- програма, която симулира движение на различни животни в изследване на биологични видове**
 - Разглеждаме класове *Riba*, *Jaba* и *Ptica*. Да се представим, че всеки от тях разширява (е производен клас на) базовия клас *Zivotno*, който съдържа метод *move()* и реализира преместването на животното в термините на координатната система *x-y*.
 - Всеки производен клас дефинира метода *move*. Нашата програма поддържа масив от референции към различни производни класове на клас *Zivotno*
 - За анимация на движението на различните животни нашата програма изпълнява метода *move* на всеки от обектите на производните класове *Riba*, *Jaba* и *Ptica*. Но всеки от тези обекти има специфичен начин на движение и всеки от елементите на масива трябва да изпълнява метода *move* на обекта, който този елемент реферира
 - Така програмата изпълнява един и същ метод *move*, а обектът рефериран от елемента на масива с животни трябва да знае как точно да промени положението в координатната система *x-y*.
 - Основавайки се на това съответния обект да знае как да “извърши правилно действието” при изпълнение на един и същ метод лежи в основата на концепцията за полиморфизъм.
 - Един и същи метод (в този случай, *move*) се изпълнява от серия обекти и има различен начин на реализация “много- форми” на реализация и отгук “*полиморфизъм*”

10.2 Примери за полиморфизъм

- Примери в химията- йерархия (*Химичен елемент- Химично съединение*) и метод *react()*, задаващ реакцията на химичното съединение на киселина (*например*)
 - На променлива от типа на базовия клас присвояваме обект от производен клас (производния клас е базовия клас!)
 - Когато програмата извиква метод, рефериран чрез променлива от типа на базовия клас, се изпълнява метода на производния клас, определен от обекта от производен клас присвоен на тази променлива
 - Същият метод (по име и “подпис”) на базовия клас води до различни форми на изпълнение, определен от обекта от производен клас присвоен на променлива от типа на базовия клас
 - Полиморфизмът позволява еднообразно обработване на нови (производни) класове без промяна на базовия код

Software Engineering факт 10.1

Полиморфизмът позволява на програмистите да моделират общо поведение за класове в йерархия от наследственост и позволи да се реализират специфичните особености на тези класове да се проявят в процеса на изпълнение на програмата

10.3 Демонстриране на полиморфично поведение

- Референция от типа на базовия клас може да вземе за стойност обект от производен клас- **преобразуване “нагоре” (upcasting)** по йерархията от наследственост
 - Това е възможно, защото обектът от производния клас **E (is-A)** също така обект от базовия клас
 - При изпълнение на метод посредством такава референция методът, който се изпълнява се определя от типа на обекта присвоен на тази референция
- Референция от производен клас може да вземе за стойност обект на базов клас само ако обектът на базовия клас се преобразува явно до производния клас- т.е. Възможно е да се осъществи **преобразуване “надолу” (downcasting)** по йерархията от наследственост

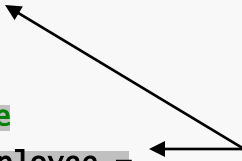
Outline

PolymorphismTest .java

(1 от 2)

```
1 // Fig. 10.1: PolymorphismTest.java
2 // Assigning superclass and subclass references to superclass and
3 // subclass variables.
4
5 public class PolymorphismTest
6 {
7     public static void main( String args[] )
8     {
9         // assign superclass reference to superclass variable
10        CommissionEmployee3 commissionEmployee = new CommissionEmployee3(
11            "Sue", "Jones", "222-22-2222", 10000, .06 );
12
13        // assign subclass reference to subclass variable
14        BasePlusCommissionEmployee4 basePlusCommissionEmployee =
15            new BasePlusCommissionEmployee4(
16                "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
17
18        // invoke toString on superclass object using superclass variable
19        System.out.printf( "%s %s:\n\n%s\n\n",
20            "Call CommissionEmployee3's toString with superclass reference ",
21            "to superclass object", commissionEmployee.toString() );
22
23        // invoke toString on subclass object using subclass variable
24        System.out.printf( "%s %s:\n\n%s\n\n",
25            "Call BasePlusCommissionEmployee4's toString with subclass",
26            "reference to subclass object",
27            basePlusCommissionEmployee.toString() );
28    }
```

Типични присвоявания за
референции



Outline

```

29 // invoke toString on subclass object using superclass variable
30 CommissionEmployee3 commissionEmployee2 =
31     basePlusCommissionEmployee;
32 System.out.printf( "%s %s:\n\n%s\n",
33     "Call BasePlusCommissionEmployee4's toString with superclass",
34     "reference to subclass object", commissionEmployee2.toString() );
35 } // end main
36 } // end class PolymorphismTest

```

Call CommissionEmployee3's toString with superclass reference to superclass object:

```

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

```

Присвоява обект от производния клас **basePlusCommissionEmployee** на референция към базов клас **CommissionEmployee3**

Call BasePlusCommissionEmployee4's toString with subclass reference to subclass object:

```

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00

```

Call BasePlusCommissionEmployee4's toString with superclass reference to subclass object:

```

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00

```

PolymorphismTest

.java

(2 от 2)

Полиморфично изпълнение на метод **toString()** на клас **basePlusCommissionEmployee** чрез референция към базовия клас (**upcasting**)



10.4 Абстрактни класове и методи

- Абстрактни класове

- Класове с твърде обща дефиниция, не позволяваща реализиране на изпълнимо поведение и създаване на обекти например *class Shape*, *class Human*, *class Animal* и пр.
- Използват се като абстрактни базови класове, стоящи в началото на йерархия от наследственост с цел маркиране на общо поведение на производните класове и данни за характеризиране на текущото състояние на обектите от тези производни класове
- Повечето йерархии от наследственост използват абстрактни класове в първите няколко (най- горни) нива от йерархията

- Конкретен клас- всеки клас, който не е абстрактен

- Ключовата дума **abstract**
 - Използва се за деклариране на **абстрактен клас**
 - Използва се за деклариране на **абстрактен метод**
 - Абстрактните класове имат **един или повече абстрактни методи**
 - Всеки от **конкретните** производни класове **ТРЯБВА** да **предефинира** всеки от абстрактните методи на **директния базов абстрактен клас**

Software Engineering факт 10.3

Всеки **абстрактен клас** декларира **общи данни и поведение** за обектите на производните класове в йерархия от наследственост.

Всеки от абстрактните методи трябва да се предефинират в първия производен конкретен клас. **Пропускането на предефиниране** на един абстрактен метод в производен клас води до изискването този клас да се третира също като абстрактен.

Данните на абстрактен клас се подчиняват на същите правила за наследственост, както в общия случай на наследственост.

Обичайна грешка при програмиране 10.1

Всеки опит да се създаде обект от абстрактен клас води до грешка при компилация.

Обичайна грешка при програмиране 10.2

Пропускане на предефиниране на абстрактен метод от директен абстрактен базов клас изисква производния клас да се декларира като `abstract`.

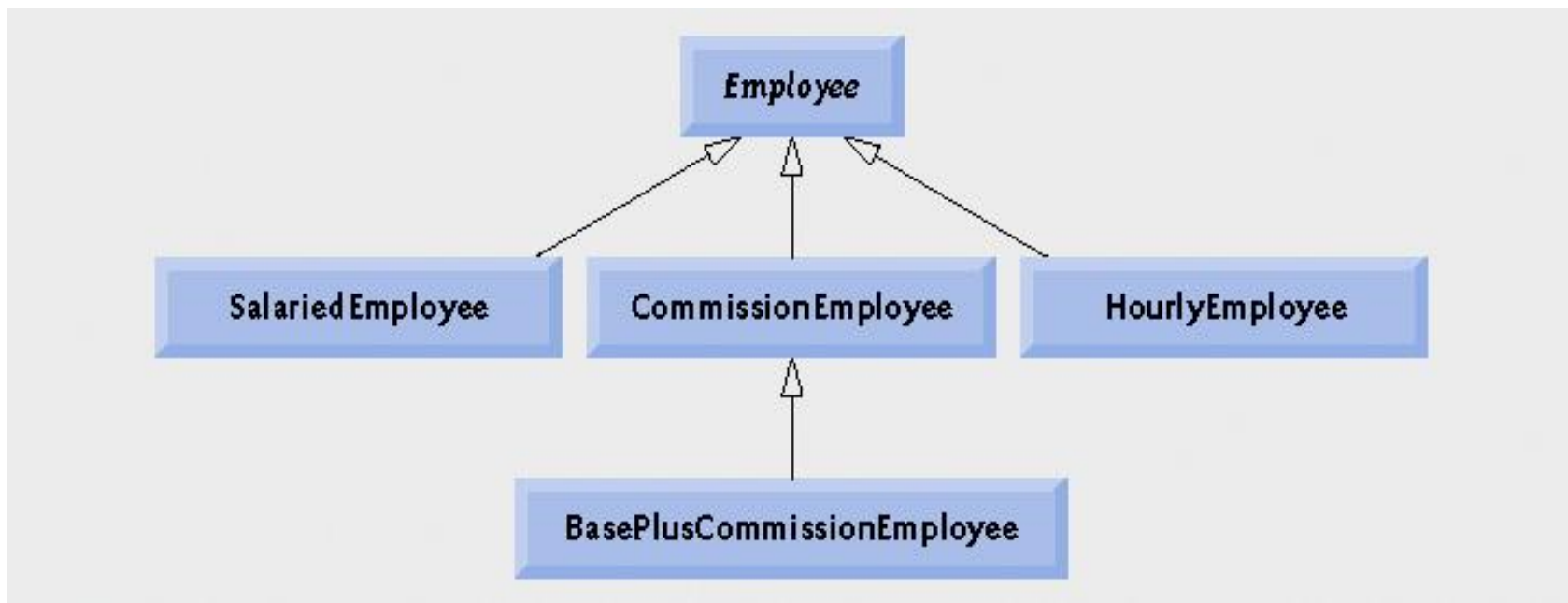


Fig. 10.2 | Employee йерархия с UML class диаграма.

10.5.1 Дефиниране на базов абстрактен клас `Employee`

- Декларираме абстрактен базов клас `abstract class Employee`
 - Декларираме метод `earnings()` като `abstract`
 - Не се задава дефиниция за метод `earnings()`
 - На това ниво от йерархията не е ясно как да се смята заплата (`earnings`) – **маркира се общо поведение**, което ще се конкретизира в производните класове
 - **`class Employee`** се декларира като абстрактен понеже **съдържа абстрактен метод**
 - Масив от `Employee` референции ще съдържа обекти на производните класове (*виж следващата диаграма*)
 - Изпълнението на метод `earnings()` чрез тези референции води до изпълнение на “правилната” версия на метода, характерна за обекта от производния клас рефериран от съответния елемент на масива—> **полиморфизъм!**

	earnings	toString
Employee	abstract	<i>firstName lastName</i> social security number: <i>SSN</i>
Salaried- Employee	weeklySalary	salaried employee: <i>firstName lastName</i> social security number: <i>SSN</i> weekly salary: <i>weeklySalary</i>
Hourly- Employee	<i>If hours <= 40</i> <i>wage * hours</i> <i>If hours > 40</i> <i>40 * wage +</i> <i>(hours - 40) * wage * 1.5</i>	hourly employee: <i>firstName lastName</i> social security number: <i>SSN</i> hourly wage: <i>wage</i> ; hours worked: <i>hours</i>
Commission- Employee	<i>commissionRate * grossSales</i>	commission employee: <i>firstName lastName</i> social security number: <i>SSN</i> gross sales: <i>grossSales</i> ; commission rate: <i>commissionRate</i>
BasePlus- Commission- Employee	<i>(commissionRate * grossSales) + baseSalary</i>	base salaried commission employee: <i>firstName lastName</i> social security number: <i>SSN</i> gross sales: <i>grossSales</i> ; commission rate: <i>commissionRate</i> ; base salary: <i>baseSalary</i>

Fig. 10.3 | Полиморфично поведение за производните класове на class Employee.

Outline

```
1 // Fig. 10.4: Employee.java
2 // Employee abstract superclass.
3
4 public abstract class Employee
5 {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9
10    // three-argument constructor
11    public Employee( String first, String last, String ssn )
12    {
13        firstName = first;
14        lastName = last;
15        socialSecurityNumber = ssn;
16    } // end three-argument Employee constructor
17
```

Декларираме **abstract** class
Employee

Данни общи за всеки от
производните класове

Employee.java

(1 от 3)



Outline

Employee.java

(2 от 3)

```
18 // set first name
19 public void setFirstName( String first )
20 {
21     firstName = first;
22 } // end method setFirstName
23
24 // return first name
25 public String getFirstName()
26 {
27     return firstName;
28 } // end method getFirstName
29
30 // set last name
31 public void setLastName( String last )
32 {
33     lastName = last;
34 } // end method setLastName
35
36 // return last name
37 public String getLastName()
38 {
39     return lastName;
40 } // end method getLastName
41
```



Outline

Employee.java

(3 от 3)

```
42 // set social security number
43 public void setSocialSecurityNumber( String ssn )
44 {
45     socialSecurityNumber = ssn; // should validate
46 } // end method setSocialSecurityNumber
47
48 // return social security number
49 public String getSocialSecurityNumber()
50 {
51     return socialSecurityNumber;
52 } // end method getSocialSecurityNumber
53
54 // return String representation of Employee object
55 public String toString()
56 {
57     return String.format( "%s %s\nsocial security number: %s",
58         getFirstName(), getLastName(), getSocialSecurityNumber() );
59 } // end method toString
60
61 // abstract method overridden by subclasses
62 public abstract double earnings(); // no implementation here
63 } // end abstract class Employee
```

Забележете:

Абстрактните методи
не трябва и нямат
дефиниция (блок от
команди за
изпълнение)

Декларация на абстрактния метод earnings

class Employee има абстрактен метод и затова
е деклариран като абстрактен клас!



Outline

```
1 // Fig. 10.5: SalariedEmployee.java
2 // SalariedEmployee class extends Employee.
3
4 public class SalariedEmployee extends Employee
5 {
6     private double weeklySalary;
7
8     // four-argument constructor
9     public SalariedEmployee( String first, String last, String ssn,
10         double salary )
11     {
12         super( first, last, ssn ); // pass to Employee constructor
13         setWeeklySalary( salary ); // validate and store salary
14     } // end four-argument SalariedEmployee constructor
15
16     // set salary
17     public void setWeeklySalary( double salary )
18     {
19         weeklySalary = salary < 0.0 ? 0.0 : salary;
20     } // end method setWeeklySalary
21
```

class **SalariedEmployee** е
произведен на class
Employee

Извиква конструктора на базовия клас!

Извиква МЕТОД **setWeeklySalary**

Проверява за валидност и присвоява
нова стойност за **weeklySalary**

SalariedEmployee
.java

(1 от 2)



Outline

```
22 // return salary
23 public double getWeeklySalary()
24 {
25     return weeklySalary;
26 } // end method getWeeklySalary
27
28 // calculate earnings; override abstract method earnings in Employee
29 public double earnings()
30 {
31     return getWeeklySalary();
32 } // end method earnings
33
34 // return String representation of SalariedEmployee object
35 public String toString()
36 {
37     return String.format( "salaried employee: %s\n%s: $%,.2f",
38         super.toString(), "weekly salary", getWeeklySalary() );
39 } // end method toString
40 } // end class SalariedEmployee
```

Предефинира (override) метод **earnings** за да може клас **SalariedEmployee** да бъде конкретен клас

Предефинира метод **toString()**

Извиква версията на **toString** от базовия клас (**макар и абстрактен!**)

SalariedEmployee

.java

(2 от 2)



Outline

HourlyEmployee
.java

(1 от 2)

```
1 // Fig. 10.6: HourlyEmployee.java
2 // HourlyEmployee class extends Employee.
3
4 public class HourlyEmployee extends Employee
5 {
6     private double wage; // wage per hour
7     private double hours; // hours worked for week
8
9     // five-argument constructor
10    public HourlyEmployee( String first, String last, String ssn,
11        double hourlyWage, double hoursWorked )
12    {
13        super( first, last, ssn );
14        setWage( hourlyWage ); // validate hourly wage
15        setHours( hoursWorked ); // validate hours worked
16    } // end five-argument HourlyEmployee constructor
17
18    // set wage
19    public void setWage( double hourlyWage )
20    {
21        wage = ( hourlyWage < 0.0 ) ? 0.0 : hourlyWage;
22    } // end method setWage
23
24    // return wage
25    public double getWage()
26    {
27        return wage;
28    } // end method getWage
29
```

class **HourlyEmployee** е
произведен на class
Employee

Извиква конструктора на базовия клас!

Проверява за валидност и
присвоява нова стойност за
hourlyWage



Outline

HourlyEmployee

.java

(2 от 2)

```

30 // set hours worked
31 public void setHours( double hoursworked )
32 {
33     hours = ( ( hoursworked >= 0.0 ) && ( hoursworked <= 168.0 ) ) ?
34         hoursworked : 0.0;
35 } // end method setHours
36
37 // return hours worked
38 public double getHours()
39 {
40     return hours;
41 } // end method getHours

```

Проверява за валидност и
присвоява нова стойност за
hoursWorked

```

42
43 // calculate earnings; override abstract method earnings in Employee
44 public double earnings()
45 {
46     if ( getHours() <= 40 ) // no overtime
47         return getWage() * getHours();
48     else
49         return 40 * getWage() + ( getHours() - 40 ) * getWage() * 1.5;
50 } // end method earnings

```

Предефинира (override) метод
earnings за да може клас
HourlyEmployee да бъде
конкретен клас

```

51
52 // return String representation of HourlyEmployee object
53 public String toString()
54 {
55     return String.format( "hourly employee: %s\n%s: $%,.2f; %s: $%,.2f",
56         super.toString(), "hourly wage", getWage(),
57         "hours worked", getHours() );
58 } // end method toString
59 } // end class HourlyEmployee

```

Предефинира метод
toString()

Извиква версията на
toString от базовия клас
(макар и абстрактен!)



Outline

```
1 // Fig. 10.7: CommissionEmployee.java
2 // CommissionEmployee class extends Employee.
3
4 public class CommissionEmployee extends Employee
5 {
6     private double grossSales; // gross weekly sales
7     private double commissionRate; // commission percentage
8
9     // five-argument constructor
10    public CommissionEmployee( String first, String last, String ssn,
11        double sales, double rate )
12    {
13        super( first, last, ssn );
14        setGrossSales( sales );
15        setCommissionRate( rate );
16    } // end five-argument CommissionEmployee constructor
17
18    // set commission rate
19    public void setCommissionRate( double rate )
20    {
21        commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
22    } // end method setCommissionRate
23
```

class **CommissionEmployee** е
произведен на class
Employee

Извиква конструктора
на базовия клас!

CommissionEmployee
.java

(1 от 3)

Проверява за валидност и
присвоява нова стойност за
commissionRate



Outline

CommissionEmployee
.java

(2 от 3)

```
24 // return commission rate
25 public double getCommissionRate()
26 {
27     return commissionRate;
28 } // end method getCommissionRate
29
30 // set gross sales amount
31 public void setGrossSales( double sales )
32 {
33     grossSales = ( sales < 0.0 ) ? 0.0 : sales;
34 } // end method setGrossSales
35
36 // return gross sales amount
37 public double getGrossSales()
38 {
39     return grossSales;
40 } // end method getGrossSales
41
```

Проверява за валидност и
присвоява нова стойност за
grossSales



Outline

Предефинира (override) метод **earnings** за да може клас **CommissionEmployee** да бъде конкретен клас

Предефинира метод **toString()**

Извиква версията на **toString** от базовия клас (**макар и абстрактен!**)

CommissionEmployee
.java

(3 от 3)



```
42 // calculate earnings; override abstract method earnings in Employee
43 public double earnings()
44 {
45     return getCommissionRate() * getGrossSales();
46 } // end method earnings
47
48 // return String representation of CommissionEmployee object
49 public String toString()
50 {
51     return String.format( "%s: %s\n%s: $%,.2f; %s: %.2f",
52         "commission employee", super.toString(),
53         "gross sales", getGrossSales(),
54         "commission rate", getCommissionRate() );
55 } // end method toString
56 } // end class CommissionEmployee
```

Outline

```
1 // Fig. 10.8: BasePlusCommissionEmployee.java
2 // BasePlusCommissionEmployee class extends CommissionEmployee.
3
4 public class BasePlusCommissionEmployee extends CommissionEmployee
5 {
6     private double baseSalary; // base salary per week
7
8     // six-argument constructor
9     public BasePlusCommissionEmployee( 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     } // end six-argument BasePlusCommissionEmployee constructor
15
16     // set base salary
17     public void setBaseSalary( double salary )
18     {
19         baseSalary = ( salary < 0.0 ) ? 0.0 : salary; // non-negative
20     } // end method setBaseSalary
21
```

class **BasePlusCommissionEmployee**
е произведен на class
CommissionEmployee

Извиква конструктора на базовия клас!

BasePlusCommission
Employee.java

(1 от 2)

Проверява за валидност и
присвоява нова стойност
за **baseSalary**



Outline

```
22 // return base salary
23 public double getBaseSalary()
24 {
25     return baseSalary;
26 } // end method getBaseSalary
27
28 // calculate earnings; override method earnings in CommissionEmployee
29 public double earnings()
30 {
31     return getBaseSalary() + super.earnings();
32 } // end method earnings
33
34 // return String representation of BasePlusCommissionEmployee object
35 public String toString()
36 {
37     return String.format( "%s %s; %s: $%,.2f",
38         "base-salaried", super.toString(),
39         "base salary", getBaseSalary() );
40 } // end method toString
41 } // end class BasePlusCommissionEmployee
```

Предефинира метод `earnings` от конкретния базов клас

Извиква МЕТОД `earnings` от конкретния базов клас

Предефинира метод `toString()`

Извиква версията на `toString` от базовия клас (**вече конкретен клас!**)

BasePlusCommission
Employee.java

(2 от 2)



Outline

PayrollSystemTest
.java

(1 of 5)

```
1 // Fig. 10.9: PayrollSystemTest.java
2 // Employee hierarchy test program.
3
4 public class PayrollSystemTest
5 {
6     public static void main( String args[] )
7     {
8         // create subclass objects
9         SalariedEmployee salariedEmployee =
10             new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
11         HourlyEmployee hourlyEmployee =
12             new HourlyEmployee( "Karen", "Price", "222-22-2222", 16.75, 40 );
13         CommissionEmployee commissionEmployee =
14             new CommissionEmployee(
15                 "Sue", "Jones", "333-33-3333", 10000, .06 );
16         BasePlusCommissionEmployee basePlusCommissionEmployee =
17             new BasePlusCommissionEmployee(
18                 "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );
19
20         System.out.println( "Employees processed individually:\n" );
21     }
```



Outline

PayrollSystemTest

.java

(2 от 5)

```
22 System.out.printf( "%s\n%s: $%,.2f\n\n",
23     salariedEmployee, "earned", salariedEmployee.earnings() );
24 System.out.printf( "%s\n%s: $%,.2f\n\n",
25     hourlyEmployee, "earned", hourlyEmployee.earnings() );
26 System.out.printf( "%s\n%s: $%,.2f\n\n",
27     commissionEmployee, "earned", commissionEmployee.earnings() );
28 System.out.printf( "%s\n%s: $%,.2f\n\n",
29     basePlusCommissionEmployee,
30     "earned", basePlusCommissionEmployee.earnings() );
31
32 // create four-element Employee array
33 Employee employees[] = new Employee[ 4 ];
34
35 // initialize array with Employees
36 employees[ 0 ] = salariedEmployee;
37 employees[ 1 ] = hourlyEmployee;
38 employees[ 2 ] = commissionEmployee;
39 employees[ 3 ] = basePlusCommissionEmployee;
40
41 System.out.println( "Employees processed polymorphically:\n" );
42
43 // generically process each element in array employees
44 for ( Employee currentEmployee : employees )
45 {
46     System.out.println( currentEmployee ); // invokes toString
47 }
```

Преобразуване “нагоре”
(upcasting) →
присвояване на обекти от
производни класове на
референции към базов
клас

Неявно и полиморфично извикване на метод **toString**



Outline

```

48 // determine whether element is a BasePlusCommissionEmployee
49 if ( currentEmployee instanceof BasePlusCommissionEmployee )
50 {
51     // downcast Employee reference to
52     // BasePlusCommissionEmployee reference
53     BasePlusCommissionEmployee employee =
54         ( BasePlusCommissionEmployee ) currentEmployee;
55
56     double oldBaseSalary = employee.getBaseSalary();
57     employee.setBaseSalary( 1.10 * oldBaseSalary );
58     System.out.printf(
59         "new base salary with 10%% increase is: $%,.2f\n",
60         employee.getBaseSalary() );
61 } // end if
62
63 System.out.printf(
64     "earned $%,.2f\n\n", currentEmployee.earnings() );
65 } // end for
66
67 // get type name of each object in employees array
68 for ( int j = 0; j < employees.length; j++ )
69     System.out.printf( "Employee %d is a %s\n", j,
70         employees[ j ].getClass().getName() );
71 } // end main
72 } // end class PayrollSystemTest

```

Когато **променливата** `currentEmployee` е обект от class `BasePlusCommissionEmployee`

Преобразува “**надолу**” `currentEmployee` до референция към class `BasePlusCommissionEmployee`

PayrollSystemTest
.java

(3 от 5)

Увеличава базовата заплата **само на обекти** `BasePlusCommissionEmployees` с 10%

Полиморфично извикване на метод `earnings`

Извиква `getClass` и `getName` методи и **извежда името на класа** на всеки произведен на клас `Employee`



Outline

PayrollSystemTest

.java

(4 of 5)

Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: \$800.00
earned: \$800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: \$16.75; hours worked: 40.00
earned: \$670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: \$10,000.00; commission rate: 0.06
earned: \$600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: \$5,000.00; commission rate: 0.04; base salary: \$300.00
earned: \$500.00



Outline

PayrollSystemTest

.java

(5 от 5)

Employees processed polymorphically:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: \$800.00
earned \$800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: \$16.75; hours worked: 40.00
earned \$670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: \$10,000.00; commission rate: 0.06
earned \$600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: \$5,000.00; commission rate: 0.04; base salary: \$300.00
new base salary with 10% increase is: \$330.00
earned \$530.00

Employee 0 is a SalariedEmployee
Employee 1 is a HourlyEmployee
Employee 2 is a CommissionEmployee
Employee 3 is a BasePlusCommissionEmployee

Същите резултати, както
ако обектите бяха
обработени поотделно

Базовата заплата увеличена с
10%

Извеждане на името на класа на
всеки обект от производен
клас



10.5.6 Демонстриране на полиморфизъм, оператор `instanceof` и преобразуване “*надолу*”

- **Динамично свързване**
 - Известно още като “късно свързване”
 - Извикването на предефинираните методи се отлага до времето на изпълнението на програмата и разпознаването им се основава на обекта рефериран от съответната променлива към базов клас
- **оператор `instanceof`**
 - Определя дали даден обект принадлежи към зададено име на клас

Обичайна грешка при програмиране 10.3

Присвояване на променлива към базов клас на променлива към производен клас **без явно преобразуване** към производния клас води до **грешка при компилация**.

```
public class A {}
public class B extends A{}
public class C {
    public static void main(String[] args)
    {
        A aVar = new B(); // upcasting!
        B bVar = aVar;     // грешка при downcasting!
        B bbVar = (B)aVar; // правилно направен downcasting!
    }
}
```

Software Engineering факт 10.5

По време на изпълнение е възможно да се прави преобразуване “*надолу*” на референция към директен или индиректен базов клас.

Преди да се извърши това преобразуване е препоръчително да се използва оператора **instanceof** за проверка, че референцията има стойност към обект от производния клас.

Обичайна грешка при програмиране 10.3

```
public class A {}
public class Dummy {}
public class B extends A{}
public class C {
    public static void main(String[] args)
    {
        A aVar = new B(); // upcasting!
        Dummy d= new Dummy();
        if (aVar instanceof B) // проверка ПРЕДИ downcasting!
        {
            B bbVar = (B)aVar; // правилно направен downcasting!
            B erVar = (B)d;     // ClassCastException!
        }
    }
}
```

Обичайна грешка при програмиране 10.4

При преобразуване надолу се извежда прекъсване по изключение **ClassCastException** когато преобразувания обект не е в *is-a* връзка на наследственост с типа на преобразуването.

10.5.6 Демонстриране на полиморфизъм, оператор `instanceof` и преобразуване “*надолу*” ...

- Преобразуване “*надолу*”
- Преобразува референция към базов клас до референция към производен клас
 - Разрешена е само когато референцията към базовия клас е инициализирана с обект, който е в “*is-a*” релация на наследственост с производния клас
- метод `getClass`
 - Онаследен от `class Object`
 - Връща обект от `class Class`
- метод `getName` на `class Class`
 - Връща името на класа

10.5.7 Обобщение на позволените присвоявания между променливи, рефериращи базов и производен клас

- Правила за присвояване на референции от базов и производен клас
 - Присвояването на референция към обект от производен клас на референция към базов клас се извършва непосредствено с неявно преобразуване (преобразуване нагоре/upcasting)
 - Присвояването на референция към обект от базов клас на референция към производен клас се извършва с явно преобразуване (преобразуване надолу/downcasting)
 - Преобразуването нагоре е “**сигурно**” защото е ясна наследствената връзка
 - Реферирането на членове, характерни само за производния клас, чрез референция към базов клас **води до грешка** при компилация
 - Директно извършване на преобразуване надолу (без явно преобразуване до производния клас) води до грешка при компилация
 - Използва се оператора *instanceof* за откриване на невъзможност за явно преобразуване

Задачи

1. Напишете един *abstract class MyBase* , който няма методи.
 - Напишете *class MyDerived* произведен на *class MyBase* и нека *class MyDerived* да има някакъв нестатичен метод *void method()* За определеност нека този метод извежда съобщение, че метод с това име “*method*” се изпълнява от обект на клас *class MyDerived*
 - Напишете още *static* метод *void myStaticMethod(MyBase bRef)* в *class MyDerived*, който да преобразува надолу *bRef* до *class MyDerived* и извиква *method()* За определеност нека този метод *myStaticMethod* извежда съобщение, че метод с това име “*myStaticMethod*” се изпълнява от клас *class MyDerived*
 - Накрая, напишете *main()* метод в *class MyDerived* и демонстрирайте, че метод *myStaticMethod(MyBase bRef)* работи.
2. Напишете, нова версия на *class MyBase*, където добавите *void method()* като абстрактен метод и покажете, че и без нужда от преобразуване надолу метод *void myStaticMethod(MyBase bRef)* работи, както преди.