НАСЛЕДЯВАНЕ, ПОЛИМОРФИЗЪМ И ИНТЕРФЕЙСИ

гл.ас. д-р Мария Евтимова

https://github.com/marias83837/JavaPresentations

mevtimova@tu-sofia.bg

Наследяване

- процес при ,който един клас придобива свойствата на друг клас
- информацията е управляема в йерархичен ред

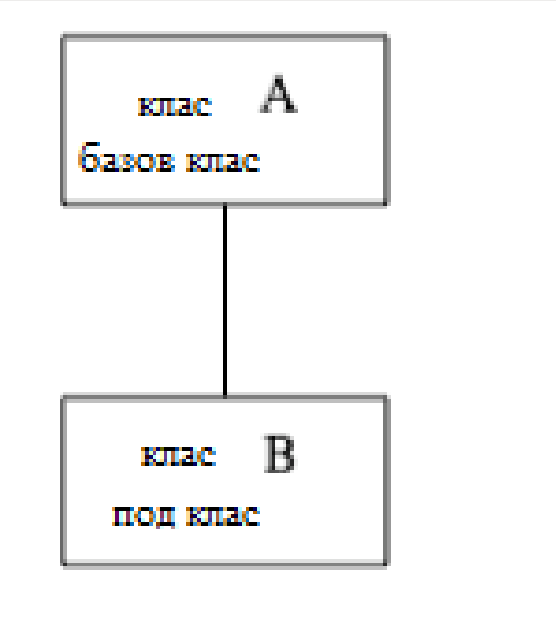
Синтаксис:

class SuperClass{ .... }

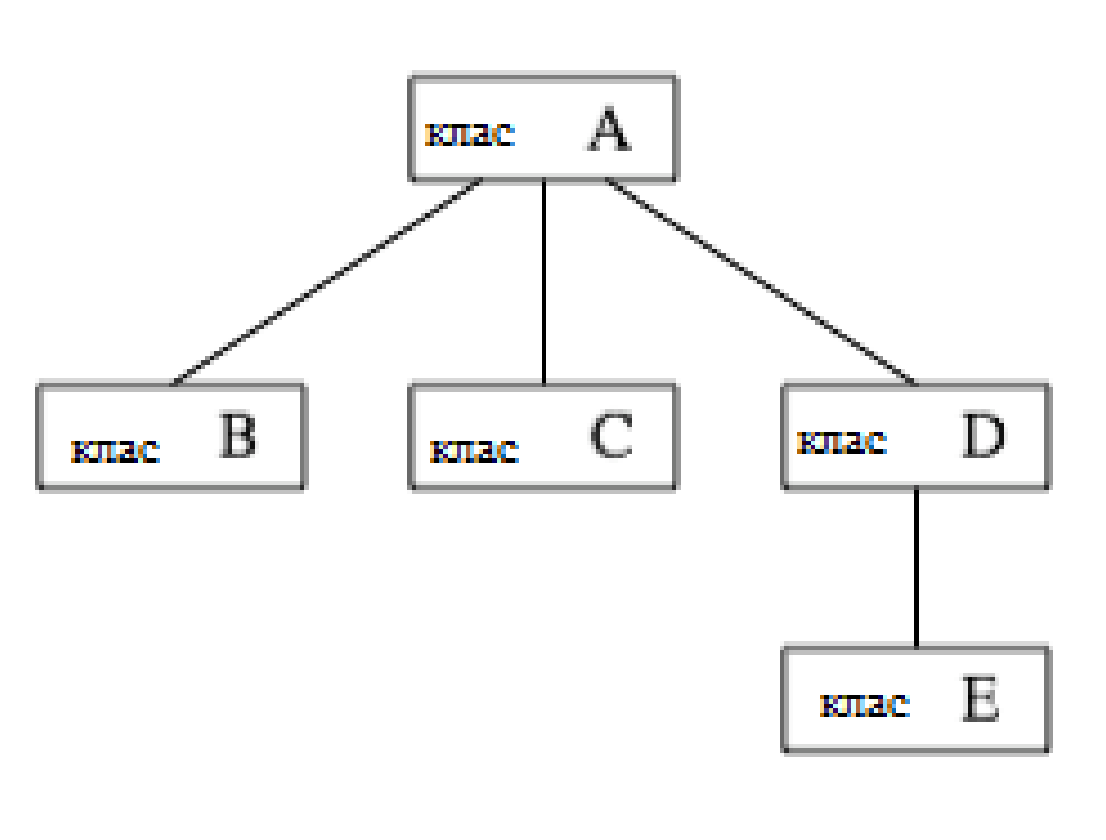
class SubClass **extends** SuperClass { .... }

подклас- клас, който наследява свойствата на друг (производен клас, клас на дете)

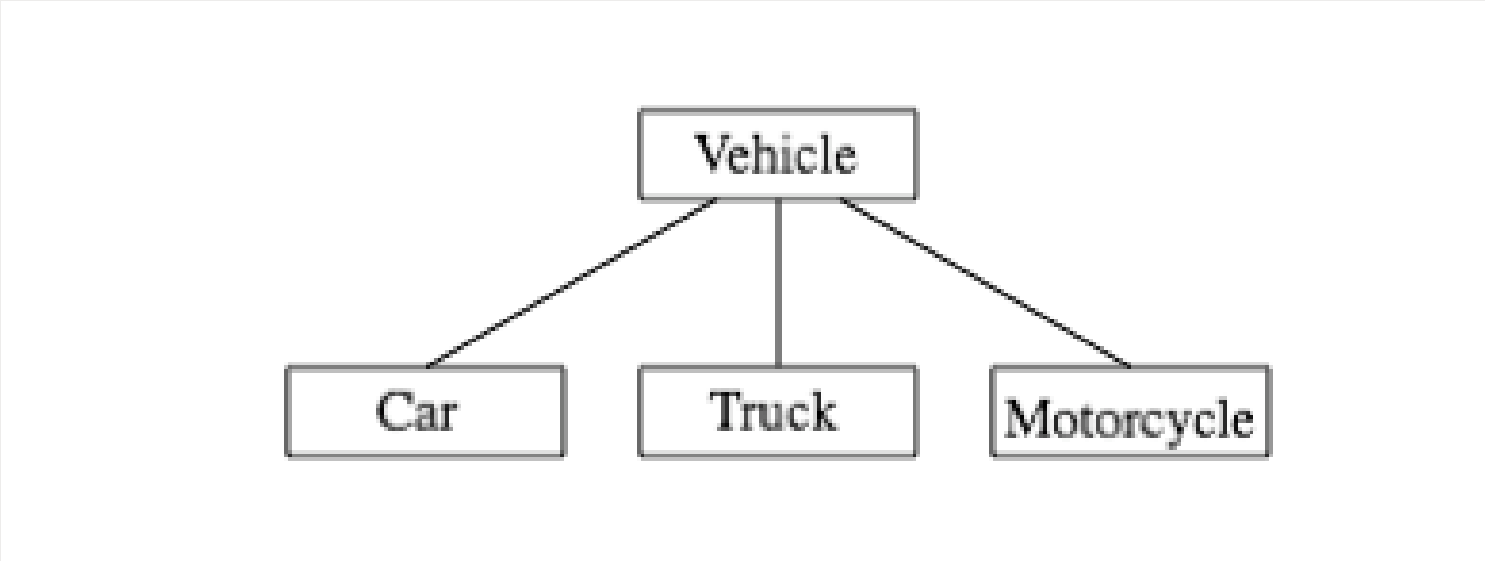
суперклас- класът, чиито свойства са наследени (базов, родителски)



няколко класа, като подкласа



Пример за моторни превозни средства



Правила за наследяване

- **конструкторите не са членове на класа и не се наследяват**
- **подкласът наследява всички достъпни членове на базовия клас с изключение на скритите от него полета и предефинираните от него методи**

Достъп до членовете и конструктори на базовия клас се осъществява чрез ключовата дума super

Правила за наследяване

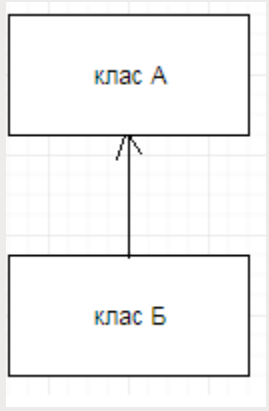
- наследява **public** и **protected** членовете на директния си супер клас
- наследява членовете без спецификатор за достъп (достъп по подразбиране), ако са в един и същ пакет
- не наследява член или метод със същото име
- не наследява **private** членовете на базовия клас

Видове наследственост

- единствена наследственост

public class A{.....}

public class B extends A{.....}

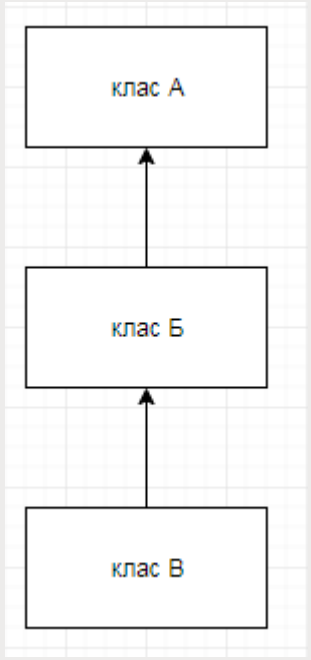


- наследственост на много нива

public class A{.....}

public class B extends A{ .....}

public class C extends B{.....}



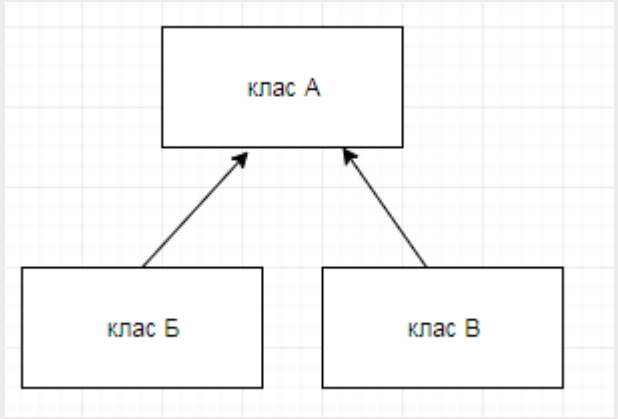
Видове наследственост

- йерархична наследственост

public class A{.....}

public class B extends A{.....}

public class C extends A{.....)

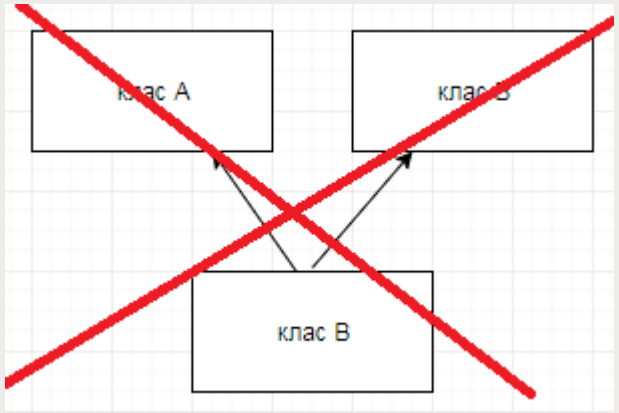


- множествена наследственост

public class A{.....}

public class B{.....}

public class C extends A,B{.....}



```
class Vehicle {
int registrationNumber;
Person owner;
void transferOwnership(Person newOwner) {
 . . .
 }
. . . }
 class Car extends Vehicle {
int numberOfDoors;
. . . }
 class Truck extends Vehicle {
 int numberOfAxles;
. . .
}
 class Motorcycle extends Vehicle {
 boolean hasSidecar;
 . . . }
```

```
class BasicArithmetic {
float z;
 public void addition(int x, int y) {
 z = x + y;
System.out.println("The sum of the given numbers:"+z); }
public void subtraction(int x, int y) {
 z = x - y;
 System.out.println("The difference between the given numbers:"+z);
} }
 public class Add_Calculation extends BasicArithmetic {
 public void multiplication(int x, int y) {
z = x * y;
 System.out.println("The product of the given numbers:"+z);
}
 public void division(int x, int y) {
z = x / y;
 System.out.println("The division of the given numbers:"+z);
}
public static void main(String args[]) {
int a = 10, b = 5;
Add_Calculation calculation = new Add_Calculation();
 calculation.addition(a, b);
 calculation.subtraction(a, b);
 calculation.multiplication(a, b);
 calculation.division(a,b);

} }
```

Резултати

```
The sum of the given numbers:15.0
The difference between the given numbers:5.0
The product of the given numbers:50.0
The division of the given numbers:2.0
```

```java
class SuperClass {
    int num=5;
    // извежда съдържанието на супер класа
    public void print() {
        System.out.println("Това е метод от базов клас");
    }
}
public class SubClass extends SuperClass {
    int num = 10;

    // извежда съдържанието на под класа
    public void print() {
        System.out.println("Това е метод от под клас");
    }
    public void method() {
        // създаване на обект от под класа
        SubClass sub = new SubClass();
        // извиква print() метода на под класа
        sub.print();
        // извиква print() метода на базовия клас
        super.print();
        // извежда стойността на променливата num от под класа
        System.out.println("стойност на променливата num в под класа:"+ sub.num);
        // извежда стойността на променливата num от базовия клас
        System.out.println("стойност на променливата num в базовия клас:"+ super
            .num);
    }
    public static void main(String args[]) {
        SubClass obj = new SubClass();
        obj.method();
    }
}
```

Резултати

```
Това е метод от под клас
Това е метод от базов клас
стойност на променливата num в под класа:10
стойност на променливата num в базовия клас:5
```

Връзка IS- A

- този обект е тип на този обект

```
public class Algorithms {

}

public class DataBase  extends Algorithms {

}

public class DistributedSystem extends
Algorithms {

}

public class DataBaseSearch extends
DataBase {

}
```

```java
class Algorithms {
}

class DataBase extends Algorithms {
}

 class DistributedSystem extends Algorithms {
}

public class SearchDataBase extends DataBase {
    public static void main(String args[]) {
      Algorithms a = new Algorithms();
      DataBase d = new DataBase();
      SearchDataBase s = new SearchDataBase();

      System.out.println(d instanceof Algorithms);
      System.out.println(s instanceof DataBase);
      System.out.println(s instanceof Algorithms);
   }

}
```

Резултат от програмата

```
true
true
true
```

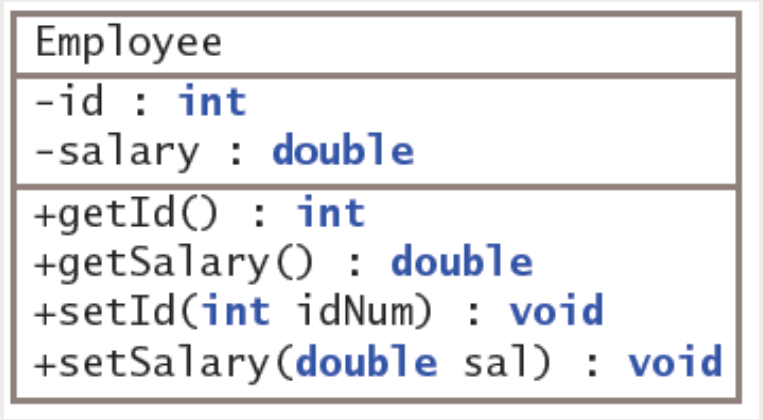
Връзка HAS- A

- определя дали даден клас има определено нещо
- взаимоотношенията се основават главно на използването
- тази връзка помага да се намали дублирането на код, както и на грешки

Пример

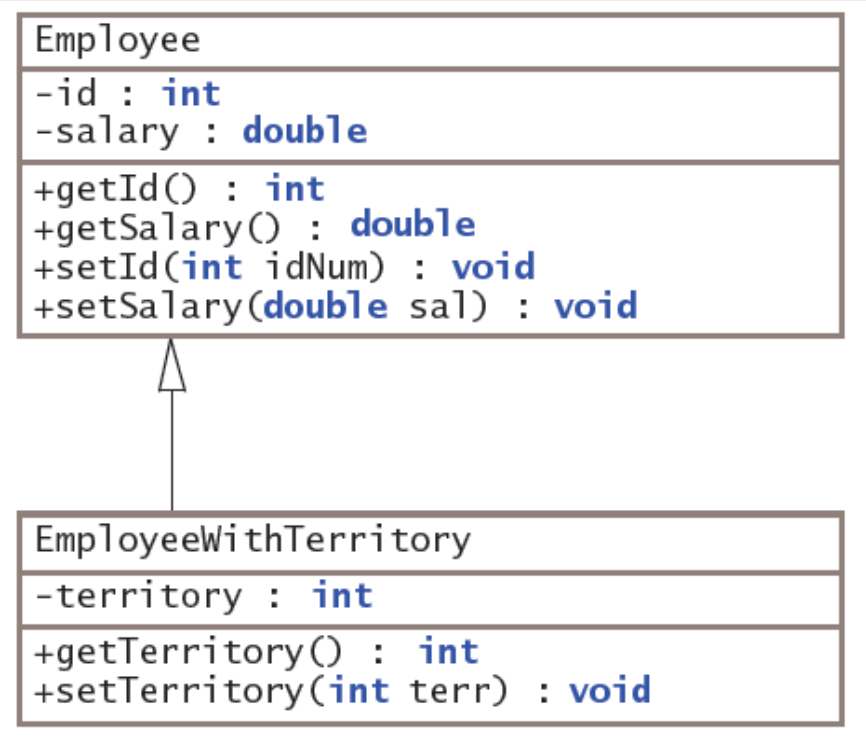
```
public class Vehicle{
}
 public class Power{
}
 public class Car extends Vehicle {
 private Power  p;
 }
```

Създаване на UML диаграма на клас




```
public class Employee
{
   private int id;
   private double salary;
   public int getId()
   {
      return id;
   }
   public double getSalary()
   {
      return salary;
   }
   public void setId(int idNum)
   {
      id = idNum;
   }
   public void setSalary(double sal)
   {
      salary = sal;
   }
}
```

Създаване на UML диаграма при наследяването




```
public class EmployeeWithTerritory extends Employee
{
   private int territory;
   public int getTerritory()
   {
      return territory;
   }
   public void setTerritory(int terr)
   {
      territory = terr;
   }
}
```

Единствения възможен начин за достъп до обект е чрез референтна променлива

- референтната променлива може да бъде само от един тип
- референтната променлива може да бъде пренасочена към други обекти, при условие че не е обявена за окончателна
- една референтна променлива може да се отнася до всеки обект от неговия деклариран тип или всеки подтип от неговия деклариран тип.

Пример

public interface **Mammal**{}

public class **Animal**{}

public class **Dog** extends **Animal** implements **Mammal**{}

Класа Dog е полиморфен, защото има множествена наследственост

IS- A връзки

Dog **IS-A** Animal

Dog **IS-A** Mammal

Dog **IS-A** Dog

Dog **IS-A** Object

Прилагане на факти за референтни променливи към обектната референция на Dog

- **Dog d= new Dog();**
- **Animal a= d;**
- **Mammal m= d;**
- **Object o=d;**

Начин на свързване на обекти (binding)

- **предварително свързване (early binding)–** процеса на връзка с тялото на даден метод при неговото извикване преди стартирането на програмата
- **късно свързване(late binding)-** процеса на свързване се извършва по време на изпълнението на програмата и се базира на типа на обекта

Създаване на полиморфни класове

- метод в подклас, поддържащ полиморфизъм, предефинира метод в супер класа;
- методът на подкласа се извиква чрез референция на супер класа- компилаторът определя типа на действителния обект по време на изпълнение и извиква подходящия метод в подкласа, от който обектът е създаден

Видове полиморфизъм

- чрез наследяване;
- чрез абстрактни класове;
- чрез интерфейси;

Пример за полиморфизъм

```
public class Shape{

 public void draw(){
System.out.println(“Draw figure”)

}
public class Square extends Shape{
 ...
 @Override
public void draw(){
System.out.println(“ square”);
 } }
public class  Triangle extends Shape{
...
@Override public void draw(){
System.out.println(“triangle");
 } }
```

Извикване на виртуален метод (Virtual method invocation)

Подкласовете на даден клас могат да дефинират свои собствени уникални поведения и да споделят някои от същите функции на родителския клас.

```
public class MountainBike extends Bicycle {
    private String suspension;

    public MountainBike(
               int startCadence,
               int startSpeed,
               int startGear,
               String suspensionType){
        super(startCadence,
              startSpeed,
              startGear);
        this.setSuspension(suspensionType);
    }

    public String getSuspension(){
      return this.suspension;
    }

    public void setSuspension(String suspensionType) {
        this.suspension = suspensionType;
    }

    public void printDescription() {
        super.printDescription();
        System.out.println("The " + "MountainBike has a" +
            getSuspension() + " suspension.");
    }
}
```

```
public class RoadBike extends Bicycle{
    // In millimeters (mm)
    private int tireWidth;

    public RoadBike(int startCadence,
                    int startSpeed,
                    int startGear,
                    int newTireWidth){
        super(startCadence,
              startSpeed,
              startGear);
        this.setTireWidth(newTireWidth);
    }

    public int getTireWidth(){
      return this.tireWidth;
    }

    public void setTireWidth(int newTireWidth){
        this.tireWidth = newTireWidth;
    }

    public void printDescription(){
        super.printDescription();
        System.out.println("The RoadBike" + " has " + getTireWidth() +
            " MM tires.");
    }
}
```

```
public class TestBikes {
  public static void main(String[] args){
    Bicycle bike01, bike02, bike03;

    bike01 = new Bicycle(20, 10, 1);
    bike02 = new MountainBike(20, 10, 5, "Dual");
    bike03 = new RoadBike(40, 20, 8, 23);

    bike01.printDescription();
    bike02.printDescription();
    bike03.printDescription();
  }
}
```

Виртуалната машина на Java (JVM) извиква подходящия метод за обекта, който е посочен във всяка променлива.
Той не извиква метода, определен от типа на променливата.

```
Bike is in gear 1 with a cadence of 20 and travelling at a speed of 10.

Bike is in gear 5 with a cadence of 20 and travelling at a speed of 10.
The MountainBike has a Dual suspension.

Bike is in gear 8 with a cadence of 40 and travelling at a speed of 20.
The RoadBike has 23 MM tires.
```

Скриване на променливи (hiding)

-променли на подклас, която има **същото име като на променлива в супер класа**, дори ако техните типове са различни, скрива променливата в супер класа;

-достъп до скрита променлива:

super.<име_на_скрита_променлива>

-не се препоръчва използване на скрити променливи–кодът се чете трудно.

Абстрактен клас- **abstract**

- абстрактните класове могат или не могат да съдържат абстрактни методи, т.е. методи без тяло
- класът е абстрактен, ако в един клас има поне един абстрактен метод
- ако даден клас е обявен за абстрактен, той не може да бъде инстанция (екземпляр).
- за да използвате абстрактен клас, трябва да го наследите от друг клас, да осигурите реализации на абстрактните методи в него.
- ако наследите абстрактен клас, трябва да предоставите реализации на всички абстрактни методи в него.

```
abstract class Employee {

private String name;

private String position;

 private int id;

public Employee(String name, String position, int
id) {

     System.out.println("Създаване на служител");

    this.name = name;

   this.position = position;

  this.id = id;

 }

 public double computeSalary() {

    System.out.println("Изчисляване на заплатата");

  return 0.0;    }

    public void sendPay() {

System.out.println("Изплащане на заплатата " +
this.name + " " + this.position);

 }

   public String toString() {

  return name + " " + position + " " + id;

 }

public String getName() {

   return name;

 }

    public String getPosition() {

     return position;

  }

 public void setPosition(String newPosition) {
position = newPosition;

  }

   public int getId() {

  return id;    }}
```

```
public class AbstractExample {
  public static void main(String [] args) {
     /* Това не е позволено и дава грешка */
    Employee e = new Employee("Петър В.", "Техник", 43);
   System.out.println("\n Изплащане на заплата на служител--");
    e.sendPay();
}}
```

```
AbstractExample.java:46: error: Employee is abstract; cannot be instantiated
      Employee e = new Employee("Петър В.", "Техник", 43);
                   ^
1 error
```

```
public class Salary extends Employee
{
 private double salary;   // Годишна заплата
 public Salary(String name, String position, int id, double salary) {
     super(name, position, id);
    setSalary(salary);    }
     public void sendPay() {
  System.out.println("Send payment ");
  System.out.println("Send payment to " + getName() + " with salary " + salary);
  }
public double getSalary() {
  return salary;
 }
   public void setSalary(double newSalary) {
      if(newSalary >= 0.0) {
      salary = newSalary;
  }   }
  public double computePay() {
  System.out.println("Computing salary pay for " + getName());
   return salary/52;
}}
```

```
public class AbstractExample {
public static void main(String [] args) {
    Salary s = new Salary("Ivan", "Engineer", 3, 3600.00);
    Employee e = new Salary("Petar Ivanov", "Seller", 2, 2400.00);
   System.out.println("Call send payment using Salary reference --");
    s.sendPay();
    System.out.println("\n Call send payment using Employee
reference--");
    e.sendPay();
  }
}
```

Резултат

Не може да се създаде инстанция на класа Employee, но може да се създаде инстанция на класа Salary

```
Employee
Employee
Call send payment using Salary reference --
Send payment
Send payment to Ivan with salary 3600.0

 Call send payment using Employee reference--
Send payment
Send payment to Petar Ivanov with salary 2400.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

достъп до 3 полета и 7 метода на класа Employee

Абстрактни методи

- abstract се използва за обявяване на метода, като абстрактен
- abstract се използва преди името на метода в декларация за метода
- един абстрактен няма тяло
- метода има ;

Пример за абстрактен метод

```
public abstract class Employee {
private String name;
private String  position;
private int id;
public abstract double computePay();
}
```

Последствия от използването на абстрактен метод

- класът, който го съдържа, трябва да бъде обявен като абстрактен
- всеки клас, който наследява текущия клас, трябва или да замени абстрактния метод, или да се обяви за абстрактен

Интерфейси

- Средство за взаимодействие между обекти, като между тях няма взаимни връзки
- Наименована съвкупност от дефиниции на методи (без реализации) и декларации на константи

Всеки клас, който иска да използва даден интерфейс, трябва да реализира всичките му методи

Интерфейс и клас

- Интерфейсът може да съдържа всякакъв брой методи
- Интерфейсът се записва във файл с разширение .java, като името на интерфейса съответства на името на файла
- Байтовият код на интерфейса се появява в .class файл
- Интерфейсите се появяват в пакети и съответния им байт код трябва да бъде в структура на директория, която съответства на името на пакета

Интерфейс ≠ клас

- Не можете да създадете интерфейс
- Интерфейсът не съдържа конструктори
- Всички методи в интерфейса са абстрактни
- Интерфейсът не може да съдържа полета за потребителски модели. Единствените полета, които могат да се появяват в интерфейса, трябва да бъдат декларирани и като static и като final
- Интерфейсът не се разширява от клас, той се изпълнява от клас
- Един интерфейс може да разшири няколко интерфейса

Интерфейс ≠ абстрактен клас

- интерфейсът и абстрактния клас не са еквивалентни;
- Интерфейсът е списък от нереализирани и следователно абстрактни методи
- Ако даден клас използва абстрактен клас, трябва да наследи абстрактния клас, но ако вече има супер клас, трябва да използва интерфейса

Java не поддържа множествено наследяване

Свойства на интерфейса

- Интерфейсът е имплицитно абстрактен. Не е необходимо да използвате ключовата дума abstract, докато декларирате интерфейс.
- Всеки метод в интерфейса е също имплицитно абстрактен и абстрактната ключова дума не е необходима
- Методите в интерфейса са имплицитно публични

Деклариране на интерфейс

```
interface Animal {
 public void eat();
 public void sleep();
 }
```

Имплементиране на интерфейс в класа

```
public class Mammal implements Animal {
public void eat() {
 System.out.println("Mammal eats");
 }
public void sleep() {
System.out.println("Mammal sleeps");
 }
public int noOfLegs() {
return 0;
}
 public static void main(String args[])
{
Mammal m = new Mammal();
m.eat();
 m.sleep();
} }
```

Резултат

Mammal eats

Mammal sleeps

Правила за имплементиране на интерфейс

- Един клас може да реализира повече от един интерфейс в даден момент
- Един клас може да разшири само един клас, но реализира много интерфейси
- Един интерфейс може да разшири друг интерфейс, по подобен начин като клас може да разшири друг клас

Множествено наследяване на интерфейси

Интерфейсите и множественото наследяване са различни понятия:

- Класът наследява от интерфейса само константи;
- Класът не наследява от интерфейса реализации на методи;
- Йерархията на интерфейсите е независима от йерархията на класовете;

Класовете, които реализират един и същ интерфейс, могат да бъдат или да не бъдат свързани чрез йерархия на класовете; това не е вярно за множественото наследяване

Java позволява множествено наследяване на интерфейси, т.е. един интерфейс може да има много супер интерфейси.

Приложение на интерфейсите

Интерфейсът се използва да дефинира **протокол за поведение**, който може да се реализира от произволен клас на произволно място в йерархията на класовете.

- Обединява сходствата между несвързани класове без изкуствено налагане на връзки между тях;
- Декларира методи, които могат да бъдат реализирани от един или повече класове;
- Разкрива програмния интерфейс на обектите, без да разкрива самите класове (т.н. анонимни обекти, които са полезни при изпращане на пакет от класове на друг проектант)