Интерфейси

Интерфейс

- **средство за взаимодействие между обекти, като между тях няма взаимни връзки;**
- **аналогични са на протоколи.**

1. Определение

Интерфейсът е 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.

2. Разлика между интерфейс и абстрактен клас

- **интерфейсът и абстрактният 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 интерфейс (Java не поддържа множествено наследяване).**

Интерфейсите се използват от класове, между които няма задължителни връзки – те просто реализират специфични методи.

3. Множествено наследяване на интерфейси

Интерфейсите и множественото наследяване са различни способи:

- **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 същи интерфейс, могат да бъдат или да не бъдат свързани чрез йерархията 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.

4. 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 сходство между несвързани класове без изкуствено налагане на връзки между тях;**
- **декларира методи, които се предполага 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).**

5. Дефиниране на интерфейс

<модификатор_на_интерфейс> interface <идентификатор> extends <списък_от_имена_на_супер_интерфейси> <тяло_на_интерфейс>

<списък_от_имена_на_супер_интерфейси> - <име_на_супер_интерфейс>, ...

<тяло_на_интерфейс> -

```
{
    <декларации на константи>;
    <декларации на методи>;
}
```

Интерфейсът може да наследява много интерфейси.

Класът може да наследява само един директен супер клас.

<модификатор_на_интерфейс> - | public

достъп по подразбиране – **за класовете от същия пакет;**

public – **достъпен за произволен клас от произволен пакет.**

<декларации_на_константи> – **неявно са** public, static **и** final**.**

<декларации_на_методи> – **неявно са** public **и** abstract**.**

Член на интерфейс не може да бъде private**,** protected**,** transient**,** volatile **и** synchronized**.**

6. Реализация на интерфейс

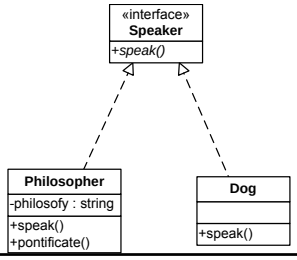
- **клас, който използва интерфейс, трябва да реализира всичките му методи;**
- **клауза** implements **в декларацията на класа:**

implements <списък_от_имена_на_интерфейси>

7. Използване на интерфейс като тип

- **декларира тип данна от сложен тип и може да се декларира променлива от тип на интерфейса;**
- **променливата може да се обръща към всеки обект от клас, който реализира този интерфейс.**

Пример: Интерфейс Speaker **(**Оратор**) с един метод** speak **(**говоря**). Интерфейсът се реализира от класа** Philosopher **(**Философ**), който добавя метода** pontificate **(**говоря надуто**) и от класа** Dog **(**Куче**).**



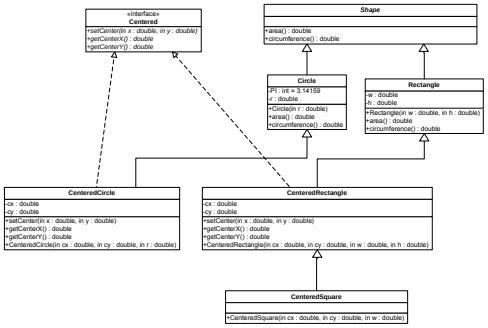

```
// Декларация на интерфейс Оратор
public interface Speaker {
  public void speak();
}
// Клас Философ реализира интерфейса Оратор
public class Philosopher implements Speaker {
  private String philosophy;
  public Philosopher(String thoughts) {
    philosophy = thoughts;
  }
  public void speak() {
    System.out.println(philosophy);
  }
  public void pontificate() {
    for (int i = 1; i <= 3; i++)
        System.out.println(philosophy);
  }
}
```

```
// Клас Куче реализира интерфейса Оратор
public class Dog implements Speaker {
  public void speak() {
    System.out.println("Джаф-джаф");
  }
}
public class Talking {
  public static void main(String[] args) {
    Speaker current;              // декларира променлива от тип Speaker
    current = new Dog();          // референция към обект от класа Dog
    current.speak();              // извиква метода speak на Dog
    current = new Philosopher("Аз мисля, следователно съществувам.");
                                  // референция към обект от класа Philosopher
    current.speak();              // извиква метода speak на Philosopher
    ((Philosopher)current).pontificate();  // pontificate не е метод на Speaker –
                                  // необходимо е преобразуване до типа на Philosopher
  }
}
```

Резултати:

```
Джаф-джаф
Аз мисля, следователно съществувам.
Аз мисля, следователно съществувам.
Аз мисля, следователно съществувам.
Аз мисля, следователно съществувам.
```

Пример: Реализация на йерархията:




```
public interface Centered {                    // интерфейс Центриран
  public void setCenter (double x, double y);  // методи на интерфейса
  public double getCenterX();
  public double getCenterY();
}
public abstract class Shape {                  // абстрактен клас Фигура
  public abstract double area();               // абстрактен метод лице
  public abstract double circumference();      // абстрактен метод периметър
}
public class Circle extends Shape {            // Окръжност наследява Фигура
  private static final double PI = 3.14159;
  private double r;
  public Circle (double r) {  this.r = r;  }
  public double area () {  return PI * r * r;  }          // реализира методите
  public double circumference () {  return 2 * PI * r;  } // на Фигура
}
public class Rectangle extends Shape {   // Правоъгълник наследява Фигура
  private double w, h;                    // широчина, височина
  public Rectangle (double w, double h) {  this.w = w;  this.h = h;  }
  public double area() {  return w * h;  }              // реализира методите
  public double circumference () {  return 2 * (w + h);  } // на Фигура
}
```

```
public class CenteredCircle extends Circle implements Centered {
  private double cx, cy;
  public CenteredCircle(double cx, double cy, double r) {
    super(r);
    this.cx = cx;
    this.cy = cy;
  }
  public void setCenter(double x, double y) {  cx = x;  cy = y;  }
  public double getCenterX() {  return cx;  }
  public double getCenterY() {  return cy;  }
}
public class CenteredRectangle extends Rectangle implements Centered {
  private double cx, cy;
  public CenteredRectangle(double cx, double cy, double w, double h) {
    super(w,h);
    this.cx = cx;
    this.cy = cy;
  }
  public void setCenter(double x, double y) {  cx = x; cy = y;  }
  public double getCenterX() {  return cx;  }
  public double getCenterY() {  return cy;  }
}
```

```
public class CenteredSquare extends CenteredRectangle {
  public CenteredSquare(double cx, double cy, double w) {
    super(cx, cy, w, w);
  }
}
public class TestInterface {
  public static void main(String[] args) {
    Shape[] shapes = new Shape[3];
    shapes[0] = new CenteredCircle(1.0, 10.0, 1.0);
    shapes[1] = new CenteredSquare(2.0, 5.0, 3.0);
    shapes[2] = new CenteredRectangle(2.0, 4.0, 3.0, 4.0);
```

```
    double totalArea = 0.0;
    double totalDistance = 0.0;
    for (int i = 0; i < shapes.length; i++) {
      totalArea += shapes[i].area();
      if (shapes[i] instanceof Centered) {
        Centered c = (Centered)shapes[i];
        double cx = c.getCenterX();
        double cy = c.getCenterY();
        totalDistance += Math.sqrt(cx * cx + cy * cy);
      }
    }
    System.out.println("Средно лице = " + totalArea / shapes.length);
    System.out.println("Средно разстояние = " +
                                          totalDistance / shapes.length);
  }
}
```

Полиморфизъм

Полиморфизъм означава „имащ много форми“.

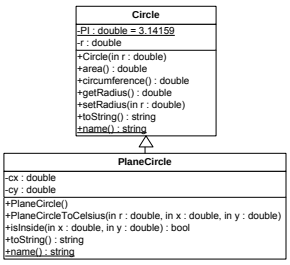
1. **Определение – техника, която позволява на класовете да осигурят различни реализации на методи с еднакви имена. Изпълнителната система извиква версията на метода според използвания обект.**

   Неполиморфните класове използват ранно (статично) свързване (early binding) – извиква се реализацията на метода на базовия клас за всеки обект.

   Полиморфните класове използват късно (динамично) свързване (late binding) – компилаторът избира версията на метода по време на изпълнение.

2. **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**
3. **Видове полиморфизъм**
   - **чрез наследяване;**
   - **чрез абстрактни класове;**
   - **чрез интерфейси.**

Полиморфизъм чрез наследяване



```
public class Circle {                                   // Окръжност
  ...
  public String toString() {  return "Окръжност с радиус " + r;  }
}
public class PlaneCircle extends Circle{                // Кръг наследява Окръжност
  ...
  public String toString() {
    return "Кръг с център (" + cx + ", " + cy + ") и радиус " + getRadius();
  }
}
public class TestCircle {
  public static void main(String[] args) {
    Circle circle = new PlaneCircle();
    System.out.println(circle);
  }
}
```

Метод toString в базовия клас Circle

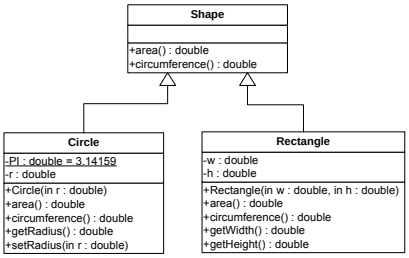
Предефинира toString на Circle

Референция на супер класа Circle се свързва с обект от тип PlaneCircle

Извиква метода toString на подкласа PlaneCircle

```
Кръг с център (0.0, 0.0) и радиус 1.0
```

Полиморфизъм чрез абстрактни класове



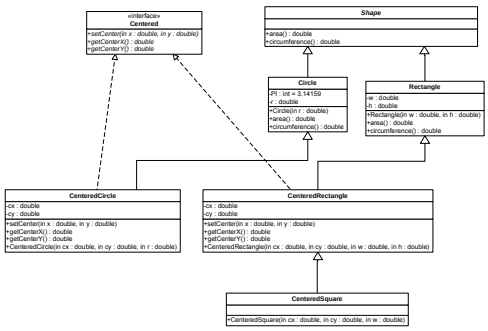

```
public abstract class Shape {                    // абстрактен клас Фигура
  public abstract double area();                 // абстрактен метод лице
  public abstract double circumference();        // абстрактен метод периметър
}
public class Circle extends Shape {              // Окръжност наследява Фигура
  ...
  public double area () {  return PI * r * r;  }          // реализира методите
  public double circumference () {  return 2 * PI * r;  } // на Фигура
}
public class Rectangle extends Shape {    // Правоъгълник наследява Фигура
  ...
  public double area() {  return w * h;  }                // реализира методите
  public double circumference () {  return 2 * (w + h);  } // на Фигура
}

Shape shape;
shape = new Circle (2.0);
shape.area();
shape = new Rectangle (1.0, 3.0);
shape.area();
```

Извиква метода area на подкласа Circle

Извиква метода area на подкласа Rectangle

Полиморфизъм чрез интерфейси




```
public interface Centered {                      // интерфейс Центриран
  public void setCenter (double x, double y);    // методи на интерфейса
  public double getCenterX();
  public double getCenterY();
}
public abstract class Shape {                    // абстрактен клас Фигура
  ...
}
public class Circle extends Shape {              // Окръжност наследява Фигура
  ...
}
public class Rectangle extends Shape {    // Правоъгълник наследява Фигура
  ...
}
public class CenteredCircle extends Circle implements Centered {
  ...
  public void setCenter(double x, double y) {  cx = x;  cy = y;  }
  public double getCenterX() {  return cx;  }
  public double getCenterY() {  return cy;  }
}
```

```
public class CenteredRectangle extends Rectangle implements Centered {
  ...
  public void setCenter(double x, double y) {  cx = x; cy = y;  }
  public double getCenterX() {  return cx;  }
  public double getCenterY() {  return cy;  }
}
public class CenteredSquare extends CenteredRectangle {
  ...
}

Centered center;
double x, y;
center = new CenteredCircle(1.0, 10.0, 1.0);
x = center.getCenterX(); y = center.getCenterY();
System.out.println("Център (" + x + ", " + y + ")");          // Център (1.0, 10.0)
center = new CenteredSquare(2.0, 5.0, 3.0);
x = center.getCenterX(); y = center.getCenterY();
System.out.println("Център (" + x + ", " + y + ")");          // Център (2.0, 5.0)
center = new CenteredRectangle(2.0, 4.0, 3.0, 4.0);
x = center.getCenterX(); y = center.getCenterY();
System.out.println("Център (" + x + ", " + y + ")");          // Център (2.0, 4.0)
```