Класове и обекти

1. **Определение на обект – програмна съвкупност от променливи и свързаните с тях методи.**

   Обектът в програмата моделира обект от реалния свят и се характеризира със:
   - **състояние (представя се чрез променливи);**
   - **поведение (представя се чрез методи).**

   Обектите взаимодействат помежду си чрез изпращане на съобщения, които включват:
   - **обект, към който е адресирано съобщението;**
   - **име на използвания метод;**
   - **необходими параметри на метода.**

   Екземпляр (инстанция) (instance) на класа е един от всички обекти от този тип.

Пример:

Обект ВЕЛОСИПЕД
- **състояние (текуща скорост, две колела, брой скорости);**
- **поведение (спиране, ускорение, смяна на скорости).**

МоятВелосипед **е екземпляр на** ВЕЛОСИПЕД**.**

2. **Определение на клас – проект или прототип, който дефинира променливите и методите, които са общи за всички обекти от даден вид.**

   Променливите изграждат ядрото на проекта, а методите ограждат ядрото и го скриват, т.е. се използва капсулиране на променливите, за да скрият детайлите на реализацията.

   Пример: Превключване скоростите на велосипеда – не ни интересува как работи самия механизъм, а трябва да знаем кой лост да превключим.

   Предимства на капсулирането:
   - **модулност;**
   - **скриване на информация.**

Видове полета и методи
- **полета (променливи) и методи на екземплярите (instance fields, instance methods) – за да се използва даден клас, се създава екземпляр, т.е. обект от този тип:**
  - **системата отделя памет за променливите на екземплярите;**
  - **извикват се методите на екземплярите;**
- **полета (променливи) и методи на класа (class fields, class methods) – за да се използват тези променливи и методи, не е необходимо да се създава екземпляр на класа:**
  - **системата отделя памет еднократно за променливите на класа при първото срещане на класа в програмата;**
  - **всички екземпляри използват тази обща памет;**
  - **методите на класа оперират само с променливите на класа и нямат достъп до променливите и методите на екземплярите.**

Предимства на класовете – реализират многократно използване.

Производителят на велосипеди използва един проект, за да произведе много велосипеди.

3. **Декларация на клас**

```
<модификатор_на_клас> class <идентификатор>
                      extends <име_на_супер_клас>
                      implements <списък_от_имена_на_интерфейси> {
  <декларация на полета>
  <декларация на методи>
  <декларация на статични инициализатрори>
  <декларация на конструктори>
}
```

Ако класът е деклариран в пакет с име_на_пакет**, то пълното име на класа е** име_на_пакет.идентификатор**. Ако пакетът не е именован, пълното име на класа е** идентификатор**.**

- <модификатор_на_клас> – public | abstract | final

  public – **класът е достъпен за всички класове;**

  abstract – **незавършен клас, не може да се създаде екземпляр на абстрактен клас;**

  final – **няма подклас.**

- <декларация на полета> **(член-променливи)**

  <модификатор_на_поле> <тип> <идентификатор> = <израз>, ...
  <модификатор_на_поле> <тип> <идентификатор>[] = <инициализатор на масив>, ...

  <модификатор_на_поле> – private protected public package static final transient volatile

  private – **полето е достъпно само в класа, в който е декларирано;**

  protected – **полето е достъпно в класа, неговите подкласове и пакета, в който е деклариран;**

  public – **полето е достъпно за всички класове;**

  package – **полето е достъпно в класа и пакета;**

  static – **декларира променлива на класа, полето се инициализира, нестатично поле декларира променлива на екземплярите;**

  final – **стойността му не може да се променя, задължително се инициализира;**

  transient – **използва се при сериализирани обекти с променливо състояние;**

  volatile – **предпазва от оптимизации на компилатора.**

- <декларации на методи> – **съдържа изпълним код, предава се определен брой стойности като аргументи**

  <модификатор_на_метод> <тип_на_резултата> <идентификатор> (<списък от формални параметри>) throws списък_от_изключения> <блок> | ;

  <модификатор_на_метод> – public protected private abstract static final synchronized native

  private – **методът е достъпен само за класа;**

  protected – **методът е достъпен за класа, неговите подкласове и пакета, в който е деклариран;**

  public – **методът е достъпен за всички класове;**

  abstract – **методът не е реализиран и трябва да бъде член на абстрактен клас;**

  static – **декларира метод на класа, извиква се без обръщение към конкретен обект, не използва** this **и** super**; нестатичен метод декларира метод на екземплярите, извиква се спрямо обект, за него се отнасят** this **и** super**;**

  final – **методът не може да бъде предефиниран от своите подкласове;**

  synchronized – **използва се при работа с паралелни разклонения, които оперират върху едни и същи данни;**

  native – **методът е реализиран на друг език.**

this **и** super

this **означава стойност, която представлява указател към обекта, за който е извикан методът на екземплярите или който се конструира. Типът на** this **е класа, в който той се среща.**

super **осъществява достъп до суперкласа.**

- <декларация на статични инициализатори> – **използват се при инициализацията на клас**

  static <блок>

- <декларация на конструктори> – **конструкторът се използва при създаването на обект, който е екземпляр на класа**
  - **Конструкторът се извиква от:**
    - **израз за създаване екземпляр на клас (**new**);**
    - **метод** Instance **на класа** Class**, представящ клас;**
    - **конверсии и конкатенации от оператора** + **за конкатенация на низове;**
    - **явно извикване на други конструктори.**
  - **Конструкторът не може да се извика от метод.**
  - **Не е член на класа.**
  - **Не се наследява.**

  <модификатор_на_конструктор> <идентификатор> (<списък от формални параметри>) throws <изключение> <тяло_на_конструктор>

  <идентификатор> – **просто име на класа;**

  <модификатор_на_конструктор> – public | protected | private

  Не може да бъде abstract**,** static**,** final**,** synchronized **или** native**.**

<тяло_на_конструктор> –

{

this(<списък от аргументи>); | super(<списък от аргументи>);

<оператор>

...

}

Вика друг конструктор на същия клас

Вика конструктор на директния супер клас

Ако явно не се извиква конструктор, извиква се конструкторът без параметри на директния супер клас.

Подразбиращ се конструктор – ако не е деклариран конструктор, автоматично се създава конструктор без параметри: той извиква конструктора на супер класа без параметри и е без модификатор на достъп.

Презареждане на конструктори – в един клас може да се дефинират няколко конструктора с различен брой и различен тип параметри.

4. Създаване на обект – екземпляр на клас или масив

Стойностите на променливите на сложните типове са указатели към тези обекти.

Стойността null **означава, че променливата не сочи никъде.**

- **израз за създаване екземпляр на клас**

  new <тип_на_клас> (<списък_от_аргументи>)

  <списък_от_аргументи> – <израз>, ...

  Аргументите в списъка избират конструктор, деклариран в тялото на класа.

  <тип_на_клас> – **не може да бъде** abstract**.**

- **обръщение към членовете на клас**

  <име_на_обект>.<име_на_поле>

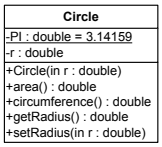
  <име_на_обект>.<име_на_метод>(<списък от аргументи>)

  super.<име_на_поле>

  super.<име_на_метод>(<списък от аргументи>)

  В Java аргументите се предават по стойност. Ако предаваният аргумент е от сложен тип, то може да се извикат неговите методи и така да се модифицират достъпните променливи в обекта.

Пример: Клас Circle **(**Окръжност**) – полета и методи на екземплярите**




```
public class Circle {                          // дефинира клас Окръжност
  private static final double PI = 3.14159;    // поле на клас – константа
  private double r;                            // поле на екземпляр – радиус
  public Circle (double r) {   this.r = r;   } // конструктор
  public double getRadius () {                 // метод на екземплярите –
    return this.r;                             // връща радиуса на
  }                                            // окръжността
  public void setRadius (double r) {           // метод на екземплярите –
    this.r = r;                                // установява радиуса на
  }                                            // окръжността
  public double area () {                      // метод на екземплярите –
    return PI * this.r * this.r;               // връща лице на окръжност
  }
  public double circumference () {             // метод на екземплярите –
    return 2 * PI * this.r;                    // връща периметър на
  }                                            // окръжност
}
```

```
public class TestCircle {                      // тестов клас
  public static void main(String[] args) {
    Circle circle = new Circle (2.0);          // създава обект
    double s = circle.area();                  // извиква метод на екземпляр
    System.out.println("Лице на окръжност = " + s);
    double p = circle.circumference();         // извиква метод на екземпляр
    System.out.println("Периметър на окръжност = " + p);
  }
}
```

Резултати:

```
Лице на окръжност = 12.56636
Периметър на окръжност = 12.56636
```

```
// Каква е стойността на радиуса на окръжността?
System.out.println(circle.r);   // Грешка! – TestCircle няма достъп до private поле r – радиусът е скрит

double radius = circle.getRadius();
System.out.println(radius);
System.out.println(circle.getRadius());

// Установяване на радиуса на окръжността
circle.r = 5.0;                 // Грешка! – TestCircle няма достъп до private поле r – радиусът е скрит

circle.setRadius(5.0);
```

Методите getRadius **и** setRadius **се наричат методи за достъп.**

```
// Печат на окръжност
System.out.println(circle);
```

Резултат:

```
Circle@4f1d0d
```

Всички класове по подразбиране наследяват класа Object **(**java.lang**).**

public String toString()

връща низ, състоящ се от името на класа, на който обектът е екземпляр, знака @ и шестнадесетичното представяне на хеш кода на обекта.

Препоръчва се всеки клас да предефинира метода toString**.**

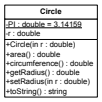
public boolean equals(Object obj)

Показва дали обекта obj **е „равен на“ този обект.**

public final Class getClass()

Връща типа на обекта.

```
public class Circle {
  …
  public String toString() {
    return "Окръжност с радиус " + this.r;
  }
}
```

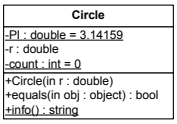



```
public class TestCircle {                          // тестов клас
  public static void main(String[] args) {
    Circle circle = new Circle (2.0);
    …
    System.out.println(circle);
  }
}
```

Резултат:

Окръжност с радиус 2.0

Пример: Клас Circle **(**Окръжност**) – полета и методи на класа**




```
public class Circle {                              // дефинира клас Окръжност
  private static final double PI = 3.14159;        // поле на клас – константа
  private static int count = 0;                    // поле на клас – брой обекти
  private double r;                                // поле на екземпляр – радиус
  public Circle (double r) {                       // конструктор
    this.r = r;
    count++;
  }
  public static void info () {                     // метод на клас
    System.out.println("Броят на окръжностите е " + count);
  }
  public boolean equals (Object obj) {             // метод на екземпляр –
    if (obj == null || getClass() != obj.getClass())  // сравнява окръжности
      return false;                                // в зависимост от техните
    Circle c = (Circle)obj;                        // радиуси
    return (r == c.r);
  }
}
```

```
public class TestCircle {
  public static void main(String[] args) {
    Circle.info();                                 // извиква статичен метод
    Circle c1 = new Circle (2.0);
    Circle c2 = new Circle (5.0);
    boolean e = c1.equals(c2);
    System.out.println("Окръжността c1 е " + (e ? "по-голяма " : "по-малка ")+
                       "от окръжността c2");
    Circle.info();                                 // извиква статичен метод
  }
}
```

Резултати:

Броят на окръжностите е 0
Окръжността c1 е по-малка от окръжността c2
Броят на окръжностите е 2

Статично поле (поле на класа)

1. Декларация

- **ключова дума** static
- **инициализира се**

```
private static int count = 0;                      // поле на клас – брой обекти
```

Статичен метод (метод на класа)

1. Декларация

- **ключова дума** static
- **използва в тялото си само статични полета (полета на класа)**

```
public static void info () {                       // метод на клас
  System.out.println("Броят на окръжностите е " + count);
}
```

3. Извиква се чрез името на класа

<име_на_клас>.<име_на_статичен_метод>(<списък от параметри>)

```
Circle.info();                                     // извиква статичен метод
```

Клас String **(**java.lang**)**

Създава символен низ, който не може да се променя.

public String(String value)

Конструира нов обект низ със същите символи като value**.**

public char charAt(int index)

Връща символа от определения index**.**

public int compareTo(String str)

Връща цяло число, показващо дали низът е лексически преди (отрицателно число), равен на (нула) или след (положително число) низа str**.**

public String concat(String str)

Връща нов низ, който се състои от низа, към който е долепен низа str**.**

public boolean equals(String str)

Връща true**, ако низът съдържа същите символи като** str **(отчита големи и малки букви) и** false **в противен случай.**

public boolean equalsIgnoreCase(String str)

Връща true**, ако низът съдържа същите символи като** str **(без да отчита големи и малки букви) и** false **в противен случай.**

public int length()

Връща броя на символите в низа.

public String replace(char oldChar, char newChar)

Връща нов низ, който е идентичен с низа, като всеки срещнат oldChar **се заменя с** newChar**.**

public String substring(int beginIndex)

Връща подниз, който започва от beginIndex **до края на низа.**

public String substring(int beginIndex, int endIndex)

Връща подниз, който започва от beginIndex **до** endIndex-1**.**

public String toLowerCase()

Връща нов низ, като всички главни букви са заменени с малки букви.

public String toUpperCase()

Връща нов низ, като всички малки букви са заменени с главни букви.

Клас StringTokenizer **(**java.util**)**

Създава низ, който се разделя на части въз основа на разделители (празна позиция, табулатор, връщане на каретката и нов ред).

public StringTokenizer(String str)

Създава нов обект StringTokenizer **низ, за да раздели низа** str **на части въз основа на разделителите по подразбиране.**

public StringTokenizer(String str, String delimiters)

Създава нов обект StringTokenizer **низ, за да раздели низа** str **на части въз основа на определените разделители** delimiters**.**

public int countTokens()

Връща броя на частите, които все още остават да бъдат обработени в низа.

public boolean hasMoreTokens()

Връща true**, ако има още части за обработка в низа.**

public String nextToken()

Връща следващата част от низа.

Пример: Брой думи в изречение

```
// Класът StringTokenizer разделя низ на части въз
// основа на определени разделители
import java.util.StringTokenizer;
public class WordCount {
  public int words (String sentence) {
    int count = 0;
    String word;
    StringTokenizer tokenizer = new StringTokenizer(sentence, " .,!?:;\t\r\n");
    while (tokenizer.hasMoreTokens()) {
      word = tokenizer.nextToken();
      System.out.println(word);
      count++;
    }
    return count;
  }
}
```

```
// Програмата въвежда последователност от
// изречения и намира броя на думите във всяко
// изречение.
import java.io.*;
public class WordCounter {
  public static void main(String[] args) throws IOException {
    BufferedReader stdin = new BufferedReader(
                           new InputStreamReader(System.in));
    String sentence;
    int count;
    WordCount counter = new WordCount();
    do {
      System.out.println("Въведете изречение:");
      sentence = stdin.readLine();
      count = counter.words(sentence);
      System.out.println("Брой думи: " + count);
      System.out.println("Ще въвеждате ли друго изречение (y/n)? ");
      sentence = stdin.readLine();
    }
    while(sentence.equalsIgnoreCase("y"));
  }
}
```

Наследяване

Наследяване е механизъм, при който даден клас наследява състоянието и поведението на всички свои родители и има възможност за разширяване.

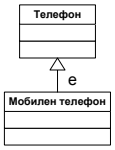
- **клаузата** extends **в декларацията на един клас определя неговия директен супер клас (родител, базов клас);**
- **самият клас е директен подклас (дете, производен клас) на класа, който разширява;**
- java.lang.Object **е класът, от който произхождат всички други класове**
  - **той няма директен суперклас;**
  - **ако в дефиницията на един клас липсва клаузата** extends**, този клас неявно е директен подклас на супер класа** Object**.**

Видове отношения

1. **Отношение е – подкласът описва множество от обекти, което се явява подмножество от обектите, описани от супер класа – използва се наследяване.**

   МОБИЛЕН ТЕЛЕФОН е ТЕЛЕФОН

   клас ТЕЛЕФОН
   набирам
   свързвам
   клас МОБИЛЕН ТЕЛЕФОН наследява ТЕЛЕФОН



2. **Отношение има – обект е композиран от други обекти – използва се композиция.**

   АВТОМОБИЛ има ДВИГАТЕЛ

   клас ДВИГАТЕЛ
   ...
   клас АВТОМОБИЛ
   ДВИГАТЕЛ двигател
   ...

1. **Декларация на клас**

   <модификатор_на_клас> class <идентификатор> extends <име_на_супер_клас>
   <тяло_на_клас>

2. **Декларация на конструктори**

- **конструкторите не се наследяват в подкласа;**
- **конструкторът на супер класа се извиква чрез ключовата дума** super**;**
- **в първия оператор в тялото на конструктора се извиква явно конструктор на директния супер клас чрез**

  super (<списък от аргументи>)
- **в противен случай неявно се извиква подразбиращият се конструктор (конструктор без параметри) на супер класа.**

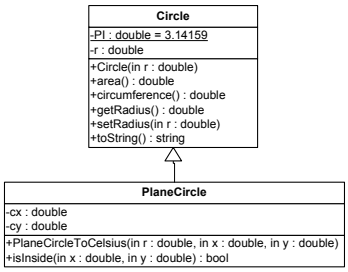
Java не поддържа множествено наследяване. Този проблем се разрешава чрез механизма на интерфейсите.

3. **Правила за наследяване**

- **конструкторите не са членове на класа и не се наследяват;**
- **подкласът наследява всички достъпни членове на директния си супер клас с изключение на скритите от него полета и предефинираните от него методи:**
  - **наследява** public **и** protected **членове на директния си супер клас;**
  - **наследява членовете без определител на достъпа (подразбиращ се достъп), ако са в един и същи пакет;**
  - **не наследява член със същото име: ако е поле, то скрива съответното поле на супер класа; ако е метод, той предефинира метода на супер класа;**
  - **не наследява** private **членовете на супер класа.**

  Достъп до членове и конструктори на супер класа се осъществява чрез ключовата дума super**.**

Пример: Клас PlaneCircle **(Кръг) наследява клас** Circle **(Окръжност)**



```
public class Circle {                                   // Окръжност
  private static final double PI = 3.14159;
  private double r;
  public Circle (double r) {  this.r = r;  }
  public double getRadius () {  return r;  }
  public void setRadius (double r) {  this.r = r;  }
  public double area () {  return PI * r * r;  }
  public double circumference () {  return 2 * PI * r;  }
  public String toString() {  return "Окръжност с радиус " + r;  }
}
public class PlaneCircle extends Circle{          // Кръг наследява Окръжност
  private double cx, cy;                                // център на кръг
  public PlaneCircle(double r, double x, double y) {    // конструктор
    super(r);                        // извиква конструктор на супер класа
    cx = x;
    cy = y;
  }
```

```
public boolean isInside (double x, double y) {    // проверка дали точка (x,y)
    double dx = x - cx;                           // лежи вътре в кръга
    double dy = y - cy;
    double distance = Math.sqrt((dx * dx + dy * dy);
    double radius = getRadius();                  // наследява getRadius
    return (distance < radius);
  }
}
public class TestCircle {
  public static void main(String[] args) {
    PlaneCircle c = new PlaneCircle (5.0, 1.0, 1.0);
    System.out.println("Лице на кръг = " + c.area());      // наследява area
    System.out.println("Обиколка на кръг = " + c.circumference());
    System.out.println("Точката (2, 5) лежи " +
                       (c.isInside(2.0, 5.0) ? "вътре в " : "вън от ") + "кръга.");
  }
}
```

Резултати:

```
Лице на кръг = 78.53975
Обиколка на кръг = 31.4159
Точката (2, 5) лежи вътре в кръга.
```

4. Сигнатура на метод

- **състои се от името на метода, броя и типовете на формалните му параметри;**
- **в един клас не могат да се декларират два метода с една и съща сигнатура.**

5. Презареждане на методи (overloading)

- **методи в един клас с еднакви имена, но с различни сигнатури.**

6. Предефиниране на методи на екземплярите (overriding)

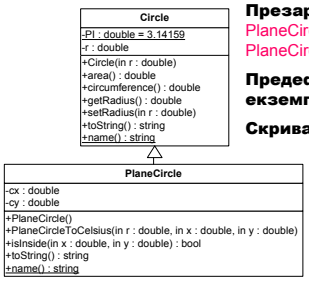
- **метод на екземплярите в подклас със същата сигнатура и върнат тип като на метод на екземплярите в супер класа, предефинира метода на супер класа;**
- **версията на извикания предефиниран метод е тази на метода на подкласа;**
- **не се предефинира статичен метод – грешка при компилация;**
- **достъп до предефиниран метод на супер класа:**

```
super.<име_на_предефиниран_метод()
```

7. Скриване на методи на класове (hiding)

- **ако подклас дефинира метод на клас със същата сигнатура като тази на метод на клас в супер класа, методът в подкласа се скрива от този в супер класа;**
- **версията на извикания скрит метод зависи дали се извиква от супер класа или от подкласа;**
- **статичен метод не може да скрива метод на екземплярите – грешка при компилация.**

Пример: Презареждане, предефиниране и скриване на методи



Презареждане на конструктори:
PlaneCircle()
PlaneCircle(double r,double x,double y)

Предефиниране на метод на екземплярите toString

Скриване на метод на класа name

```
public class Circle {                                   // Окръжност
  private static final double PI = 3.14159;
  private double r;
  public Circle (double r) {  this.r = r;  }
  public double getRadius () {  return r;  }
  public void setRadius (double r) {  this.r = r;  }
  public double area () {  return PI * r * r;  }
  public double circumference () {  return 2 * PI * r;  }
  // Метод на екземпляр
  public String toString() {  return "Окръжност с радиус " + r;  }
  // Метод на клас
  public static String name() {  return "Окръжност";  }
}
```

```
public class PlaneCircle extends Circle{          // Кръг наследява Окръжност
  private double cx, cy;                          // център на кръг
  // Конструктор без параметри (подразбиращ се конструктор)
  public PlaneCircle() {
    super(1.0);
    cx = 0;
    cy = 0;
  }
  // Конструктор с три параметъра
  public PlaneCircle(double r, double x, double y) {// конструктор  с 3 парам.
    super(r);
    cx = x;
    cy = y;
  }
  // Предефиниран метод на екземпляр
  public String toString() {
    return "Кръг с център (" + cx + ", " + cy + ") и радиус " + getRadius();
  }
  // Скрит метод на клас
  public static String name() {   return "Кръг";   }
}
```

```
public class TestCircle {
  public static void main(String[] args) {
    Circle circle = new PlaneCircle();
    System.out.println(circle.name());
    System.out.println(circle);
  }
}
```

Извиква скрития метод name **на супер класа** Circle

Извиква предефинирания метод toString **на подкласа** PlaneCircle

Резултати:

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

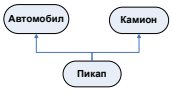
8. Скриване на променливи (hiding)

- **променлива в 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.**

8. Множествено наследяване и Java

- **Java поддържа просто наследяване – производният клас има само един родител.**

- **Някои обектно-ориентирани езици (C++) поддържат множествено наследяване – производният клас може да има много родители.**



- **Java не поддържа множествено наследяване;**
- **Използването на интерфейси дава някои от възможностите на множественото наследяване.**
- **Класът в Java наследява само един супер клас, но реализира много интерфейси.**

Абстрактни класове

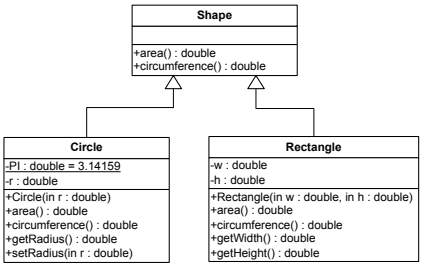
1. **Абстрактен – представя общата концепция на йерархията на класовете**
- **дефинира се с** abstract**;**
- **може да съдържа абстрактни методи – само са декларирани, но не са реализирани;**
- **не е завършен;**
- **не може да се създаде екземпляр на абстрактен клас;**
- **може да бъде наследяван;**
- **дефинира пълен програмен интерфейс, като наследниците реализират специфичните детайли на абстрактните методи.**

2. Декларация на абстрактен клас

```
abstract class <идентификатор> {
    abstract <тип_на_резултат> <идентификатор_на_ метод>
                                (<списък от формални параметри>);
  ...
}
```

3. Класът Object стои на върха на йерархията на класовете в Java.

Пример: Йерархията на класовете представя следното отношение между класовете:




```
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 getRadius () {  return r;  }
  public void setRadius (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 getWidth() {  return w;  }
  public double getHeight() {  return h;  }
  public double area() {  return w * h;  }                  // реализира методите
  public double circumference () {  return 2 * (w + h);  } // на Фигура
}
```

```
public class TestAbstract {
  public static void main(String[] args) {
    Shape[] shapes = new Shape[3];
    shapes[0] = new Circle (2.0);
    shapes[1] = new Rectangle (1.0, 3.0);
    shapes[2] = new Rectangle (4.0, 2.0);
    double total = 0.0;
    for (int i = 0; i < shapes.length; i++)
      total  += shapes[i].area();
    System.out.println("Сумата от лицата на фигурите е " + total);
  }
}
```

Вградени класове

1. Клас, който е деклариран вътре в друг клас, се нарича вграден клас.

```
class <име_на_външен_клас> {
  …
  class <име_на_вграден _клас> {
    …
  }
}
```

- **вграденият клас е член на външния клас (както променлива или метод);**
- **има достъп до променливите и методите на външния клас, дори ако те са private;**
- **произвежда отделен файл с байткод**
  <име_на_външен_клас>$<име_на_вграден_клас>.class;

- **статичен вграден клас няма достъп до променливите на екземплярите или методите, дефинирани в неговия външен клас;**
- **нестатичен вграден клас се нарича вътрешен клас**
  - **свързан е с всеки екземпляр на външния клас;**
  - **член вътре във вътрешния клас не може да бъде деклариран като static;**
  - **екземпляр на вътрешен клас може да съществува само в екземпляр на външния клас;**
  - **вътрешен клас без име се нарича анонимен клас.**

```
new <име_на_външен_клас>(<списък от параметри>)
{<тяло_на_анонимен_клас>}
```

Slide 53

MEG1 Mariana Goranova; 17.8.2006 г.