

Лекция 9.2

Класове и Обекти Част I

Основни теми

- Основни концепции на изграждане- **капсулиране** на данни и методи, **скриване на данни**
- Структуриране на данни и абстрактни типове данни
- Употреба на ключовата дума *this*.
- **Правила за структуриране на класове- данни, set и get методи, конструктори, други методи**
- Задачи

- 8.1 Въведение
 - 8.2 Пример: Моделиране на *class Time*
 - 8.3 Регулиране на правата за достъп до членовете на клас
 - 8.4 Рефериране на членовете на текущия обект посредством референцията *this*
 - 8.5 Пример: *class Time* с допълнителни дефиниции на конструктори
 - 8.6 Видове конструктори: *по подразбиране*, за “*общо ползване*” и *за копиране*
 - 8.7 Предназначение на *Set* и *Get* методите
- Задачи

Литература:

Java How to Program, Sixth Edition, глава 8

8.1 Въведение

Въведение към програмиране с класове и обекти в Java (Лекция 3.2)

- основни концепции и терминология
- методология за разработка на програми

Тази лекция разглежда:

- Принципите за **изграждане на класовете** и **ролята на членовете**, които ги съставят
- **Регулиране на достъпа** до членовете на клас
- Ролята на конструкторите на класовете и **основните видове конструктори**
- Изясняваме **принципа на композицията** при изграждане на класове
- Фокусираме внимание върху **предназначението на *set* и *get* методите**
- Въвеждаме синтаксиса и изследваме приложението на типа *enum*, позволяващ работа с **групи от константни стойности** с използване на уникални имена.
- Изясняваме връзката между *static* членове на клас и *final* променливи на обект.
- Разглеждаме приложението на **принципа за многократно използване на софтуер** посредством създаване на **библиотеки** (*package*) и връзката между класовете от една и съща библиотека (*package*) в Java.

8.2 Пример: Моделиране на *class* Time

Всеки клас капсулира данни и методи, които оперират с тези данни (Принцип на капсулирането)

Членове на клас: (Един клас “ИМА” – “HAS-A” релация)

- данни (*характеризират текущото състояние на обект от дадения клас или или класа като цяло*)

- методи (*реализират характерно поведение за всеки от обектите на даден клас или класа като цяло*)

Основни правила:

- Данните на класа трябва да са директно достъпни единствено за методите на дадения клас (**Data Hiding**)
- Данните на класа трябва да са функционално независими помежду си

8.2 Пример: Моделиране на *class* Time

Видове достъп до членовете на клас

- *public* (такива членове са общо достъпни)
- *private* (такива членове са достъпни единствено в рамките на дефиницията на класа)

Членовете на клас с *public* достъп задават *интерфейса на класа*

По правило всички клас данни са *private* (Data hiding principle)

Всяка от клас данните трябва да се *получава само валидни стойности* съобразно изискванията на задачата

8.2 Пример: Моделиране на *class* Time

Инициализацията на данните става единствено чрез специален метод- **конструктор**

При отсъствие на **явно дефиниран конструктор** компилаторът създава конструктор по подразбиране

Видове конструктори:

- **По подразбиране** - използват празен списък от аргументи
- За “**общо ползване**”- използват списък от аргументи за инициализация на всички данни на класа
- За **копиране**- използват един аргумент, който е референция към обект от дадения клас и създават друг обект от същия клас със същите стойности на данните като реферирания обект

Software Engineering факти 8.1

Всеки метод, който поменя стойностите на *private* клас данни **трябва да проверява дали новите стойности са допустими.**

Пропускането на такава проверка води до инициализация с невалидни стойности и **непредвидими последици** за изпълнението на програмата.

8.2 Пример: Моделиране на *class Time*

Пример

class Time има за цел да моделира група обекти, които да показват часовото време.

Всички обекти от този клас **ИМАТ**:

- Текущо състояние (задава данните на класа такива обекти - трябва да са *private*)
 - **Имат** “час” (валидни стойности 0- 23)
 - **Имат** “минути” (валидни стойности 0- 59)
 - **Имат** “секунди” (валидни стойности 0- 59)
- Поведение (задава методите такива обекти - трябва да са *public*)
 - Настройка на време- метод *setTime()*
 - Извеждане на текущото време- метод *toString()*

Outline

Time1.java

(1 от 2)

```
1 // Fig. 8.1: Time1.java
2 // Time1 class declaration maintains the time in 24-hour format.
3
4 public class Time1
5 {
6     private int hour;    // 0 - 23
7     private int minute;  // 0 - 59
8     private int second;  // 0 - 59
9
10    // set a new time value using universal time; ensure that
11    // the data remains consistent by setting invalid values to zero
12    public void setTime( int h, int m, int s )
13
14        hour = ( ( h >= 0 && h < 24 ) ? h : 0 );    // validate hour
15        minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); // validate minute
16        second = ( ( s >= 0 && s < 60 ) ? s : 0 ); // validate second
17    } // end method setTime
18
```

private клас данни

Декларира **public** метод **setTime**

Проверява новите стойности за валидност преди да промени клас данните



Outline

Time1.java

(2 от 2)

```
19 // convert to String in universal-time format (HH: MM: SS)
20 public String toUniversalString()
21 {
22     return String.format( "%02d: %02d: %02d", hour, minute, second );
23 } // end method toUniversalString
24
25 // convert to String in standard-time format (H: MM: SS AM or PM)
26 public String toString()
27 {
28     return String.format( "%d: %02d: %02d %s",
29         ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
30         minute, second, ( hour < 12 ? "AM" : "PM" ) );
31 } // end method toString
32 } // end class Time1
```

format на текста



8.2 Пример: Моделиране на *class Time*

- Метод *format()* на клас *String*
 - Аналогичен на метод *System.out.printf()*
 - За разлика от *printf()* метод *format()* връща форматиран текст, вместо да го извежда на **Command window**
- Ключовата дума *new* изпълнява неявно дефинирания конструктор за *class Time1*, понеже *class Time1* няма явно дефиниран такъв конструктор

Software Engineering факти 8.2

Използването на класове облекчава процеса на програмиране, понеже потребителят може да използва единствено `public` с методите на класа (задават *интерфейса* на класа).

Тези методи са повече ориентирани към потребителя, отколкото към реализацията на определено действие. Потребителите на този интерфейс не се интересуват от реализацията на това действие.

Потребителите се интересуват изключително *какво* може да прави даден клас, а **не** *как* го прави.

Software Engineering факти 8.3

Интерфейсът на даден клас се мени по- рядко от неговата реализация.

При промяна на реализацията се мени и кода, зависим от тази промяна.

Скриването на реализацията на определено поведение от потребителя изолира и ограничава евентуалните промени в рамките на класа.

Това прави потребителят на класа независим от промените в реализацията на определено поведение на този клас (Encapsulation principle!**).**

Outline

Time1Test.java

(1 от 2)

```
1 // Fig. 8.2: Time1Test.java
2 // Time1 object used in an application.
3
4 public class Time1Test
5 {
6     public static void main( String args[] )
7     {
8         // create and initialize a Time1 object
9         Time1 time = new Time1(); // invokes Time1 constructor
10
11        // output string representations of the time
12        System.out.print( "The initial universal time is: " );
13        System.out.println( time.toUniversalString() );
14        System.out.print( "The initial standard time is: " );
15        System.out.println( time.toString() );
16        System.out.println(); // output a blank line
17
```

Създава обект от клас
Time1

Изпълнява метод
toUniversalString

Изпълнява метод
toString



Outline

```

18 // change time and output updated time
19 time.setTime( 13, 27, 6 );
20 System.out.print( "Universal time after setTime is: " );
21 System.out.println( time.toUniversalString() );
22 System.out.print( "Standard time after setTime is: " );
23 System.out.println( time.toString() );
24 System.out.println(); // output a blank line
25
26 // set time with invalid values; output updated time
27 time.setTime( 99, 99, 99 );
28 System.out.println( "After attempting invalid settings: " );
29 System.out.print( "Universal time: " );
30 System.out.println( time.toUniversalString() );
31 System.out.print( "Standard time: " );
32 System.out.println( time.toString() );
33 } // end main
34 } // end class Time1Test

```

Изпълнява метод
setTime

Time1Test.java

Опит да се изпълни
МЕТОД **setTime** с
НЕВАЛИДНИ
СТОЙНОСТИ

```

The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM

Universal time after setTime is: 13:27:06
Standard time after setTime is: 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM

```



8.3 Регулиране на правата за достъп до членовете на клас

- **public** интерфейс на клас
 - Съвкупността от **public** методите определя “**услугите**”, които дадения клас предоставя на потребителите
- Реализацията на тези “**услуги**” се извършва от:
 - **private** данни и **private** методи, които са недостъпни за потребителите на класа
 - По този начин **промените в реализацията са невидими за потребителя**
 - *Потребителят не трябва да има възможност да влияе върху реализацията на тези услуги (**Data hiding!**)*

Обичайна грешка при програмиране 8.1

Опит за използване на данна или метод с *private* достъп извън дефиницията на класа води до **грешка при компилация**.

Outline

MemberAccessTest.java

```
1 // Fig. 8.3: MemberAccessTest.java
2 // Private members of class Time1 are not accessible.
3 public class MemberAccessTest
4 {
5     public static void main( String args[] )
6     {
7         Time1 time = new Time1(); // create and initialize Time1 object
8
9         time.hour = 7; // error: hour has private access in Time1
10        time.minute = 15; // error: minute has private access in Time1
11        time.second = 30; // error: second has private access in Time1
12    } // end main
13 } // end class MemberAccessTest
```

Опит за достъп до **private** данни на клас

```
MemberAccessTest.java:9: hour has private access in Time1
    time.hour = 7; // error: hour has private access in Time1
      ^
MemberAccessTest.java:10: minute has private access in Time1
    time.minute = 15; // error: minute has private access in Time1
      ^
MemberAccessTest.java:11: second has private access in Time1
    time.second = 30; // error: second has private access in Time1
      ^
3 errors
```



8.4 Рефериране на членовете на текущия обект с референцията *this*

- Референция с ключовата дума *this*
 - *this* осъществява достъп до членовете на текущия обект

Пример: *Когато локална данна в метод скрива клас данна със същото име, както локалната данна и има нужда да се реферира клас данната на текущия обект*

- Всеки не- статичен метод използва референцията *this* при рефериране на всеки от членовете си- данни и методи

Outline

```

30 // use explicit and implicit "this" to call toUniversalString
31 public String buildString()
32 {
33     return String.format( "%24s: %s\n%24s: %s",
34         "this.toUniversalString()", this.toUniversalString(),
35         "toUniversalString()", toUniversalString() );
36 } // end method buildString
37
38 // convert to String in universal-time format (HH:MM:SS)
39 public String toUniversalString()
40 {
41     // "this" is not required here to access instance variables,
42     // because method does not have local variables with same
43     // names as instance variables
44     return String.format( "%02d: %02d: %02d",
45         this.hour, this.minute, this.second );
46 } // end method toUniversalString
47 } // end class SimpleTime

```

Пример при който, метод ***toUniversalString()*** се вика явно и неявно чрез референцията ***this***

Thi sTest. j ava

(2 от 2)

Тук няма нужда от използване на ***this***

```

this.toUniversalString(): 15: 30: 19
toUniversalString(): 15: 30: 19

```



Обичайна грешка при програмиране 8.2

Честа логическа грешка се получава при използване на едно и също име за локална данна или аргумент, което се използва за клас данна

В тези случай, използването на референцията *this* е задължително за достъп до съответната клас данна.

Съвет за избягване на грешки 8.1

Избягвайте дублиране на имена на клас данни като имета на аргументи и локални данни за метод в същия клас.

Използване на ресурси 8.1

Java ограничава използваната памет като пази едно копие от всеки метод за всеки клас и този метод се вика поотделно от всеки обект на класа, когато има нужда.

От друга страна, всеки обект пази отделно копие от своите клас данни (не – статични)

Така, при изпълнението на метод се използва референцията `this` за указване кой от всички обекти на класа (и съответно клас данни) ще бъде използван при това изпълнение.

8.5 Пример: *class Time* с допълнителни дефиниции на конструктори

- Конструктор- метод специализиран за инициализация на обект при създаването му
- Конструкторът не може да се изпълнява след създаването на обекта!
- Допълнителни дефиниции на конструктори
 - Служат за осъществяване на различни способи за инициализация на обект, в съответствие с изискванията на потребителите на дадения клас
- Конструктор по подразбиране
 - Инициализира обекта с приети за стандартни начални стойности на клас данните
- референцията `this reference` може да се използва за изпълнение на друг конструктор и в този случай трябва да е първата изпълнима команда в тялото на конструктора (**software reuse!**)

Outline

Time2.java

(1 of 4)

```

1 // Fig. 8.5: Time2.java
2 // Time2 class declaration with overloaded constructors.
3
4 public class Time2
5 {
6     private int hour;    // 0 - 23
7     private int minute; // 0 - 59
8     private int second; // 0 - 59
9
10    // Time2 no-argument constructor: initializes each instance variable
11    // to zero; ensures that Time2 objects start in a consistent state
12    public Time2()
13    {
14        this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
15    } // end Time2 no-argument constructor
16
17    // Time2 constructor: hour supplied, minute and second defaulted to 0
18    public Time2( int h )
19    {
20        this( h, 0, 0 ); // invoke Time2 constructor with three arguments
21    } // end Time2 one-argument constructor
22
23    // Time2 constructor: hour and minute supplied, second defaulted to 0
24    public Time2( int h, int m )
25    {
26        this( h, m, 0 ); // invoke Time2 constructor with three arguments
27    } // end Time2 two-argument constructor
28

```

Конструктор по подразбиране

Извиква се конструкторът за общо ползване



```

29 // Time2 constructor: hour, minute and second supplied
30 public Time2( int h, int m, int s )
31 {
32     setTime( h, m, s ); // invoke setTime to validate time
33 } // end Time2 three-argument constructor

```

Конструктор за общо ползване
вика **setTime** за валидиране
на данните

```

34
35 // Time2 constructor: another Time2 object supplied
36 public Time2( Time2 time )
37 {
38     // invoke Time2 three-argument constructor
39     this( time.getHour(), time.getMinute(), time.getSecond() );
40 } // end Time2 constructor with a Time2 object argument

```

Конструктор за копиране- има за
аргумент само референция към
обект от същия клас **Time2**

```

41
42 // Set Methods
43 // set a new time value using universal time; ensure that
44 // the data remains consistent by setting invalid values to zero
45 public void setTime( int h, int m, int s )
46 {
47     setHour( h ); // set the hour
48     setMinute( m ); // set the minute
49     setSecond( s ); // set the second
50 } // end method setTime
51

```

Може да се използва и директен
достъп до данните на **time**

Outline

Time2. java

(2 от 4)



Outline

Time2.java

(3 от 4)

```
52 // validate and set hour
53 public void setHour( int h )
54 {
55     hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
56 } // end method setHour
57
58 // validate and set minute
59 public void setMinute( int m )
60 {
61     minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
62 } // end method setMinute
63
64 // validate and set second
65 public void setSecond( int s )
66 {
67     second = ( ( s >= 0 && s < 60 ) ? s : 0 );
68 } // end method setSecond
69
70 // Get Methods
71 // get hour value
72 public int getHour()
73 {
74     return hour;
75 } // end method getHour
76
```

Set и Get методи



Outline

Time2.java

(4 of 4)

```
77 // get minute value
78 public int getMinute()
79 {
80     return minute;
81 } // end method getMinute
82
83 // get second value
84 public int getSecond()
85 {
86     return second;
87 } // end method getSecond
88
89 // convert to String in universal-time format (HH:MM:SS)
90 public String toUniversalString()
91 {
92     return String.format(
93         "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
94 } // end method toUniversalString
95
96 // convert to String in standard-time format (H:MM:SS AM or PM)
97 public String toString()
98 {
99     return String.format( "%d:%02d:%02d %s",
100         ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
101         getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
102 } // end method toString
103 } // end class Time2
```



8.5 ... Правила за моделиране на клас

```
1 // Fig. 8.5: Time2.java
2 // Time2 class declaration with overloaded constructors.
3
4 public class Time2
5 {
6     private int hour;    // 0 - 23
7     private int minute; // 0 - 59
8     private int second; // 0 - 59
9 }
```

1. Пишат се данните на класа (всички с *private* достъп)

8.5 ... Правила за моделиране на клас

```

52 // validate and set hour
53 public void setHour( int h )
54 {
55     hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
56 } // end method setHour
57
58 // validate and set minute
59 public void setMinute( int m )
60 {
61     minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
62 } // end method setMinute
63
64 // validate and set second
65 public void setSecond( int s )
66 {
67     second = ( ( s >= 0 && s < 60 ) ? s : 0 );
68 } // end method setSecond
69
70 // Get Methods
71 // get hour value
72 public int getHour()
73 {
74     return hour;
75 } // end method getHour
76

```

2. Пишат се **set** и **get** методите за всички данни, които ще се проверяват за **валидност** и ще участват **public** интерфейса на класа

8.5 ... Правила за моделиране на клас

```
29  // Time2 constructor: hour, minute and second supplied
30  public Time2( int h, int m, int s )
31  {
32      setTime( h, m, s ); // invoke setTime to validate time
33  } // end Time2 three-argument constructor
```

3. Веднага след клас данните се пише конструктора за общо ползване като се използват съответните set методи за данните, гарантиращи валидност

8.5 ... Правила за моделиране на клас

```
10 // Time2 no-argument constructor: initializes each instance variable
11 // to zero; ensures that Time2 objects start in a consistent state
12 public Time2()
13 {
14     this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
15 } // end Time2 no-argument constructor
16
```

4. Веднага **след** конструктора за общо ползване се пише **конструкторът по подразбиране** като се **използва** референцията ***this*** за извикване на конструктора за общо ползване

8.5 ... Правила за моделиране на клас

```
35 // Time2 constructor: another Time2 object supplied
36 public Time2( Time2 time )
37 {
38     // Invoke Time2 three-argument constructor
39     this( time.getHour(), time.getMInute(), time.getSecond() );
40 } // end Time2 constructor with a Time2 object argument
```

5. Веднага след конструктора по подразбиране се пише **конструктора за копиране** като се използва референцията **this** за извикване на конструктора за общо ползване

8.5 ... Правила за писане на клас

```
96 // convert to String in standard-time format (H:MM:SS AM or PM)
97 public String toString()
98 {
99     return String.format( "%d:%02d:%02d %s",
100         ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
101         getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
102 } // end method toString
103} // end class Time2
```

6. Накрая се пише методът **toString()**, който служи да изведе в подходящ текстов формат текущите стойности на всички клас данни

8.5 ... Ползата от Set методите

- Използването на *set* методи в **конструктора за общо ползване**
 - Допринася за **избягване на дублиране на кода** по валидиране на данните и **опростява промени** в реализацията (**software reuse!**)

Software Engineering факти 8.4

Когато един обект на даден клас получи референция към друг обект от същия клас, то **първият обект** добива достъп до всички членове на **втория обект** (вкл. *private*)- *виж примера с сору-конструктора*

Software Engineering факти 8.5

При реализацията на методи, изискващи достъп до данните на клас, **използвайте** *set* и *get* методите на класа за достъп до `private` данните. Това облекчава поддръжката на кода и елеминира вероятността за грешки.

Обичайна грешка при програмиране 8.3

Синтактична грешка е `this` да се използва за извикване на конструктор от блока с команди на друг конструктор, ако мястото на извикване не е първата изпълнима команда в този конструктор.

Синтактична грешка е също `this` да се използва за извикване на конструктор от тялото метод, който не е конструктор.

Обичайна грешка при програмиране 8.4

Всеки конструктор може да вика други методи от същия клас. Имайте предвид, че **данните използвани от тези методи могат да не са още инициализирани**, понеже конструкторът се изпълнява в процеса на инициализация на клас данните. Използването на **данни, които не са били подходящо инициализирани** води до трудни за откриване логически грешки.

Outline

```
1 // Fig. 8.6: Time2Test.java
2 // Overloaded constructors used to initialize Time2 objects.
3
4 public class Time2Test
5 {
6     public static void main( String args[] )
7     {
8         Time2 t1 = new Time2();           // 00:00:00
9         Time2 t2 = new Time2( 2 );       // 02:00:00
10        Time2 t3 = new Time2( 21, 34 );   // 21:34:00
11        Time2 t4 = new Time2( 12, 25, 42 ); // 12:25:42
12        Time2 t5 = new Time2( 27, 74, 99 ); // 00:00:00
13        Time2 t6 = new Time2( t4 );       // 12:25:42
14
15        System.out.println( "Constructed with: " );
16        System.out.println( "t1: all arguments defaulted" );
17        System.out.printf( "    %s\n", t1.toUniversalString() );
18        System.out.printf( "    %s\n", t1.toString() );
19    }
```

Извикване на конструктори от различни дефиниции

Time2Test.java

(1 от 3)



Outline

Time2Test.java

(2 от 3)

```
20 System.out.println(  
21     "t2: hour specified; minute and second defaulted" );  
22 System.out.printf( "    %s\n", t2.toUniversalString() );  
23 System.out.printf( "    %s\n", t2.toString() );  
24  
25 System.out.println(  
26     "t3: hour and minute specified; second defaulted" );  
27 System.out.printf( "    %s\n", t3.toUniversalString() );  
28 System.out.printf( "    %s\n", t3.toString() );  
29  
30 System.out.println( "t4: hour, minute and second specified" );  
31 System.out.printf( "    %s\n", t4.toUniversalString() );  
32 System.out.printf( "    %s\n", t4.toString() );  
33  
34 System.out.println( "t5: all invalid values specified" );  
35 System.out.printf( "    %s\n", t5.toUniversalString() );  
36 System.out.printf( "    %s\n", t5.toString() );  
37
```



```
38      System.out.println( "t6: Time2 object t4 specified" );
39      System.out.printf( "    %s\n", t6.toUniversalString() );
40      System.out.printf( "    %s\n", t6.toString() );
41  } // end main
42 } // end class Time2Test
```

Outline

Time2Test.java

(3 от 3)

```
t1: all arguments defaulted
    00:00:00
    12:00:00 AM
t2: hour specified; minute and second defaulted
    02:00:00
    2:00:00 AM
t3: hour and minute specified; second defaulted
    21:34:00
    9:34:00 PM
t4: hour, minute and second specified
    12:25:42
    12:25:42 PM
t5: all invalid values specified
    00:00:00
    12:00:00 AM
t6: Time2 object t4 specified
    12:25:42
    12:25:42 PM
```



8.6 Конструктор по подразбиране

- **Инициализацията на данните на класа да става само в конструктор!**
- Всеки клас трябва да има поне един конструктор-
Препоръчително е всеки клас да има трите вида конструктори
(По подразбиране, “за общо ползване” и конструктор за копиране)
 - Ако клас няма **явно дефинирани конструктори**, компилаторът вмъква в кода дефиниция на конструктор по подразбиране)
 - В този случай данните на класа се инициализират със стойности, които са по подразбиране за съответния тип данни
 - **Примитивните типове** данни са нули за числовите типове, `false` за `boolean` променливи и `null` за всички **референтни типове данни**

Обичайна грешка при програмиране 8.5

Ако даден клас има конструктори с непразен списък от аргументи, но няма явна дефиниция за `publ i` с конструктор по подразбиране, то всеки опит да се използва конструктор по подразбиране води до грешка при компилация.

8.7 Предназначение на *Set* и *Get* методите

- *Set* методи
 - Известни също като **mutator** методи
 - Задават нови стойности на клас данни
 - Трябва да валидират новите стойности преди да инициализират клас данните
 - Може да връщат стойност, за да сигнализират невалидни данни (по принцип са **void**)
- *Get* методи
 - Известни също като **accessor** методи
 - Връщат текущата стойност на клас данни
 - Може да управляват формата на връщаните данни

Software Engineering факти 8.7

По правило, използвайте винаги `public` SET и GET методи за промяна и извличане на текущите стойности на клас данните, които пак по правило трябва да са `private`.

Тази архитектура на класа прилага основния принцип на ООП за **скриване на данните, гарантира предсказуемо поведение на обектите** от класа и **придава модулен характер** на приложението.

8.7 Предназначение на *Set* и *Get* методите

- Предикати при именуване на методи
 - Тестване дали обекта се намира в едно или друга състояние при което се очаква **boolean** резултат **true** или **false**
 - пример: един метод *isEmpty* проверява дали даден низ е инициализиран на празен низ (“”)

Задачи

1. Напишете *class Rectangle*.

- Нека *class Rectangle* има *length* (дължина) и *width* (ширина), всяка от които е 1 по подразбиране.
- Напишете SET и GET методи за *length* (дължина) и *width* (ширина). SET методът да позволява задаване на числа с плаваща запетая с единична точност от интервала [0, 20].
- Нека още обектите на този клас да имат методи за пресмятане съответно на периметъра (*calculatePerimeter*) и площта (*calculateArea*).
- Нека да има и *toString()* метод за извеждане на текущите стойности за клас данните, всяка на отделен ред със съответен промпт и 2 знака след десетичната запетая.
- Напишете приложение за тестване *class Rectangle*