

Лекция 3.2

Въведение към програмиране с
класове и обекти в Java.

Основни теми

- Деклариране на клас и създаване на обекти в Java
- Деклариране на данни на клас с оглед дефиниране на статуса на обектите на класа
- Деклариране на методи на клас с оглед дефиниране на поведението на обектите на класа
- Изпълнение на метод на обект
- Разлика между променливите за данни на клас и локалните променливи на метод
- Използване на конструктор за инициализиране данните на обект при създаването му
- Разлика между примитивни и реферетни типове данни
- Обобщение
- Задачи

- 3.1 Въведение**
- 3.2 Класове, обекти, методи и променливи на обект (инстанция на клас)**
- 3.3 Деклариране на клас един метод и създаване на обект от клас**
- 3.4 Деклариране на метод с един аргумент**
- 3.5 Променливи на обект, *set* методи и *get* методи**
- 3.6 Прimitives данни и референтни типове данни**
- 3.7 Инициализиране на обекти с конструктор**
- 3.8 Числа с плаваща запетая и тип *double***
- 3.9 Софтуерно инженерство: Идентифициране на класове при изследвана на изискванията на проблема**
- 3.10 Обобщение**
- 3.11 Задачи**

3.1 Въведение

Класове

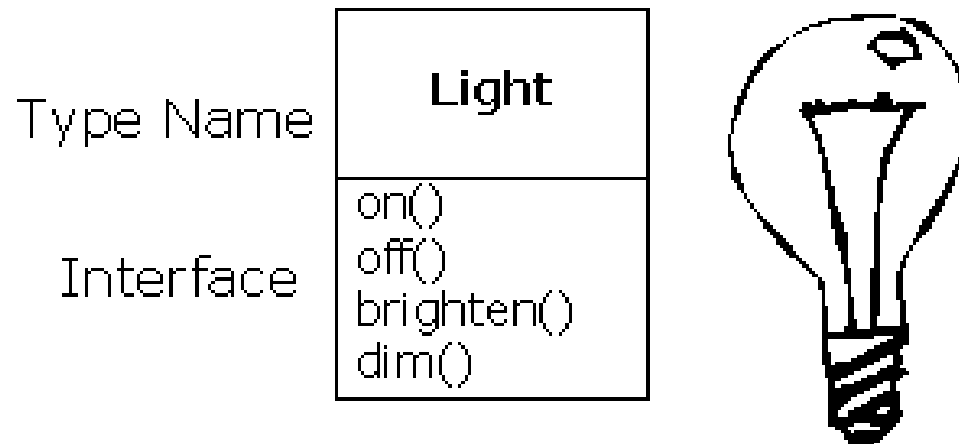
- При програмиране на Java, **основното** е да се напишат дефинициите за класовете на отделните обекти, които ще изграждат програмата.
- Всяка **дефиниция на клас** капсулира данните данните на своите обекти и тяхното поведение.
- След като класът е веднъж е дефиниран, той служи като **матрица или шаблон** за създаване на отделни обекти или т.нар инстанции от класа

3.1 Въведение

- Всеки клас съдържа **две основни групи от елементи**
 - Променливи- служат да съхраняват данни за текущото състояние (*статуса*) на обектите по отделно (*данни на обект*) или за всички обекти (*статични данни*)
 - Методи- служат да реализират определено поведение на всеки обект поотделно (*метод на обект*) или на всички обекти като цяло (статични методи)
- **За моделиране на класът**, от който произхожда даден обект трябва да се намери **отговор на следните въпроси**:
 1. **Каква роля** ще изпълнява обекта в програмата?
 2. Какви **данни определят текущото състояние** на обекта?
 3. Какво **поведение трябва да реализира обекта** съобразно данните, описващи текущото му състояние?
 4. Каква част от **поведението на обекта може да е достъпно** за съобщения (ползване) от страна на други обекти?
 5. Каква част от **поведението на обекта трябва да се скрие** и да е недостъпно за други обекти?

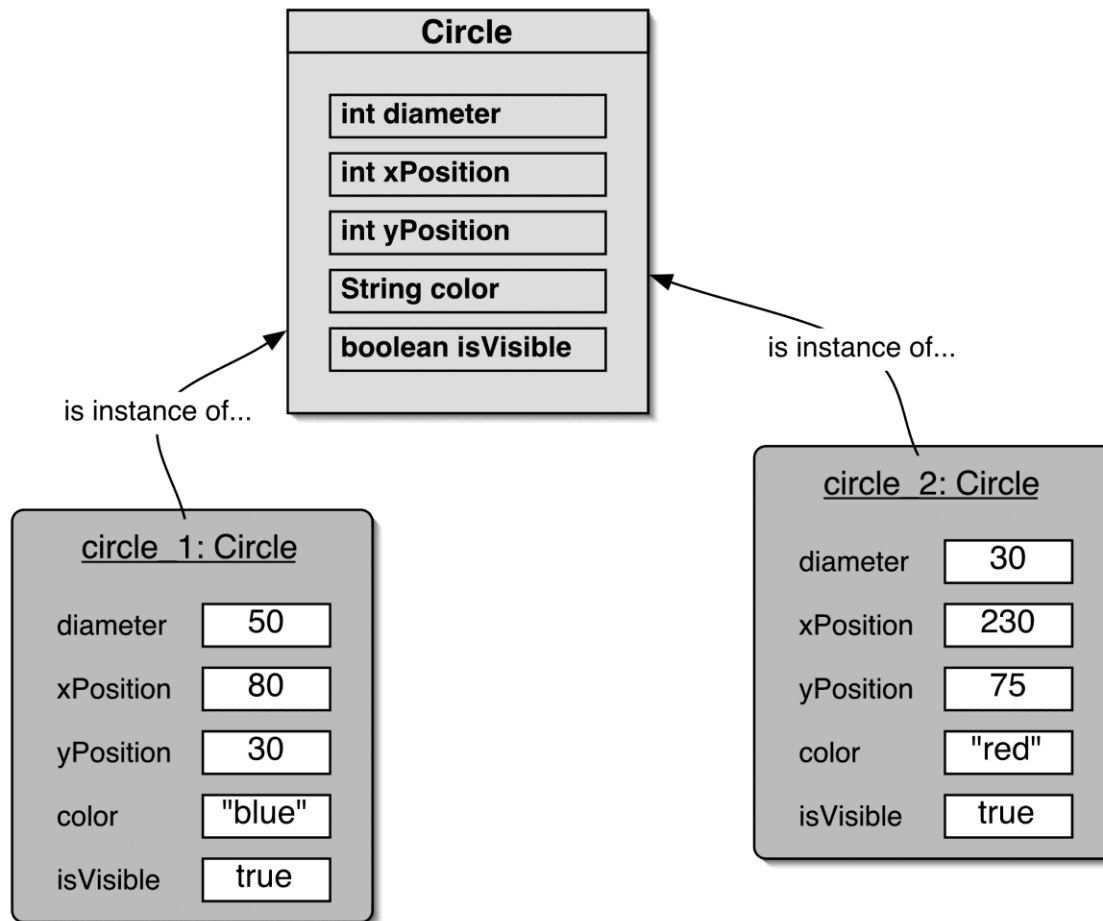
3.1 Въведение

class Light - методи за клас *Осветление*



3.1 Въведение

class Circle - данни за клас *Circle*



3.1 Въведение

Всеки клас се дефинира, чрез сорс код на Java (Java code) и описва елементите на класа (променливи `fields` и методи).

3.2 Класове, обекти, методи и променливи на обект

- Изпълнението на Java програма като управление на лека кола
- Първо някой трябва да е моделирал и после конструирал колата т.е някой е написал ***class Car*** и после създал обект ***Car***
- Само средствата за управление на колата са достъпни за шофьора т.е само част от методите реализиращи поведението на обект ***Car*** са достъпни до другите обекти в програмата
- Управлението на колата изисква със средствата за управление т.е. Изпращане на съобщения- ***messages*** до методите реализиращи поведението на обект ***Car*** (***method call***)
- Текущото състояние на колата се определя от данни, които се променят от средствата за управление- това са т. нар. данни (полета, променливи) на обекта. (***Instance variables***)
- Текущото състояние на колата може да се променя само от средствата за управление на колата- т.е. данните на обект ***Car*** могат да се променят само от методите на този обект

3.2 Класове, обекти, методи и променливи на обект

- Всеки обект съдържа едно или повече полета на данни
 - Наричат се *instance variables*
 - Те принадлежат на всеки обект от момента на създаването му и могат да се променят **единствено** от методи на същия този обект

3.2 Класове, обекти, методи и променливи на обект

Следват примери, допълващи последователно `class GradeBook` с нови елементи за демонстриране основни концепции в дефинирането на класове с Java

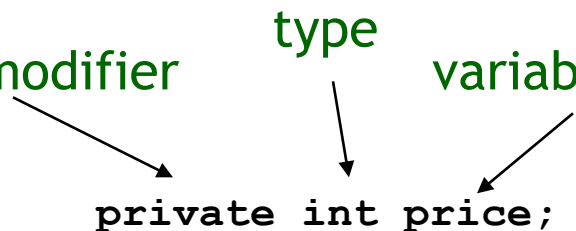
- Първи пример, дефинираме `class GradeBook` с един метод, който извежда **`welcome`** съобщение при изпълнението му. После демонстрираме как се създава обект от този клас и изпълняваме този метод, за да изведе **`welcome`** съобщение .
- Втори пример, допълва първия, позволявайки метода да получи за аргумент **`име-на-курс`** и извеждане на това име като част от **`welcome`** съобщението.
- Трети пример показва как да включим **`име-на-курс`** като данна(променлива, поле) на **`GradeBook`** обект , а също и как да използваме методи за задаване на нова стойност за **`име-на-курс`** и да получим текущата стойност на **`име-на-курс`**
- Четвърти пример демонстрира как да инициализираме данна(променлива, поле) на **`GradeBook`** обект при създаване на обекта посредством конструктора на `class GradeBook`

3.3 Деклариране на клас с един метод и създаване на обект от клас

- Декларацията на всеки клас започва с ключовите думи ***public class*** и трябва да се запише във файл със същото име, под което е дефиниран класа, следвано от окончанието ***.java*** за име на файл.

access modifier type variable name

private int price;



```
public class Car
{
    private int    price;
    private float  speed;
    private String plate;

    //пропускаме останалото...
}
```

3.3 Деклариране на клас с един метод и създаване на обект от клас

- Разглеждаме програма, състояща се от два класа *GradeBook* (**пасивен клас**) и *GradeBookTest* (**активен клас**)
- *class GradeBookTest* ще стартира изпълнението, ще създаде обект от *class GradeBook* и ще изпрати “съобщение” до метод от този клас за изпълнение на определено действие (*извежда welcome текст на Стандартен изход*)

class GradeBook

- Деклариран в отделен файл с име *GradeBook.java*
- Моделира обект *Дневник на курс* (*С какви данни и методи бихте моделирали класа Дневник на курс ? Защо това е пасивен клас?*)
- Трябва да е общо достъпен → ключова дума *public* се използва като модификатор на достъп (access modifier)
- Декларацията на всеки клас включва:
 - модификатор на достъп (access modifier)
 - Ключовата дума *class*
 - Двойка скоби - лява и дясна фигурна скоба

GradeBook.java

```
1 // Fig. 3.1: GradeBook.java
2 // Class declaration with one method.
3
4 public class GradeBook
5 {
6     // display a welcome message to the GradeBook user
7     public void displayMessage()
8     {
9         System.out.println( "welcome to the Grade Book!" );
10    } // end method displayMessage
11
12 } // end class GradeBook
```

Извежда ред от текст на Стандартен изход

Метод на *class* *GradeBook*

```
6    // display a welcome message to the GradeBook user
7    public void displayMessage()
8    {
9        System.out.println( "Welcome to the Grade Book!" );
10   } // end method displayMessage
11
```

- Използва ключова дума *public* за да укаже, че този метод е **общо достъпен**
- Използва ключова дума *void* за да укаже, че този метод **не връща данни** като резултат от изпълнението си
- Заглавието на метод (*method header*) се нарича съвкупността от модификаторът за достъп (*access modifier*), типът на връщаните данни (*return type*), името на метода и двойката фигурни скоби (лява и дясна) дефиниращи тялото на метода

Обичайна грешка при програмиране 3.1

- Декларацията на повече от един `public class` в същия файл дава грешка при компилация.

class GradeBookTest

- Служи за стартиране на Java приложението
 - съдържа *public static void main()* метод
 - създава обектите на програмата и стартира взаимодействието помежду им
 - създава обектите на програмата и стартира взаимодействието помежду им
 - *class GradeBookTest* е пример за активен клас
 - Всяко Java приложение трябва да има един такъв клас такова предназначение

class GradeBookTest

- **Java е разширяем език**
 - Програмистите могат да създават свои класове в допълнение към вече създадените библиотеки
- Израз за създаване на обект от клас
 - Използва ключовата дума *new*
 - Следвано от името на класа и кръгли скоби
- Извикване на метод на обект
 - Използва се означението *имеНаОбект.имеНаМетод()*
 - Завършва с точка и запетая
 - **Обекта трябва да е създаден, за да може да се извика метод от този обект!**

GradeBookTest.java

```
1 // Fig. 3.2: GradeBookTest.java
2 // Create a GradeBook object and call its displayMessage method.
3
4 public class GradeBookTest
5 {
6     // main method begins program execution
7     public static void main( String args[] )
8     {
9         // create a GradeBook object and assign it to myGradeBook
10        GradeBook myGradeBook = new GradeBook();
11
12        // call myGradeBook's displayMessage method
13        myGradeBook.displayMessage();
14    } // end main
15
16 } // end class GradeBookTest
```

Създава обект от class GradeBook

Извиква метод displayMessage на обект GradeBook

Welcome to the Grade Book!

Компилиране на приложение с много класове на *Command prompt window*

- **Едновременно компилиране на повече от един клас**

- Изпишете всички `.java` файлове на командния ред, разделени с празен символ

```
javac ClassOne.java ClassTwo.java  
ClassThree.java
```

- Използвайте `*.java` да компилирате всички `.java` файлове в дадената директория

```
javac *.java
```

UML Class диаграма за Class GradeBook

- **UML клас диаграми- правоъгълници, разделени вертикално на три части**
 - Горната част съдържа името на класа
 - Средната част съдържа данните на класа или т. нар. **instance variables**
 - Долната част съдържа описание на имената и типа данни за всеки метод на класа
 - Плюс знак означава **public** метод или данна
 - Минус знак означава **private** метод или данна

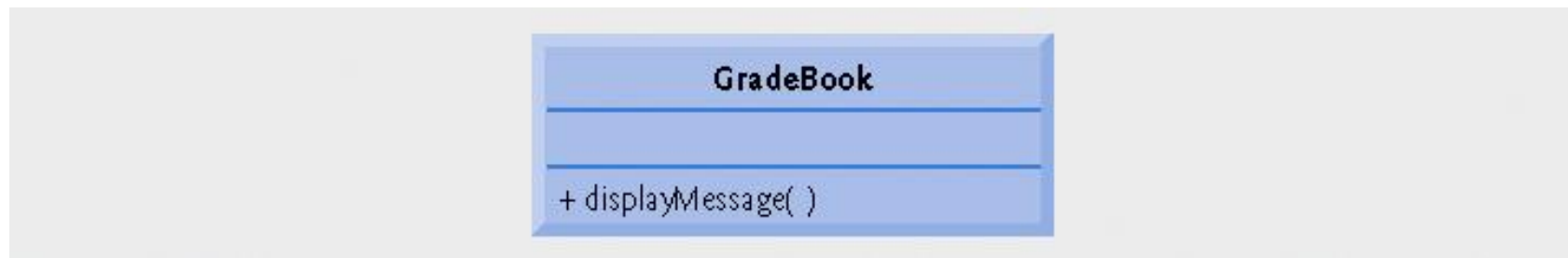


Fig. 3.3 | UML class диаграма, показваща, че *class GradeBook* има *public void displayMessage()* МЕТОД.

3.4 Деклариране на метод, използващ аргумент

- **Аргументи на методи**
 - Предават допълнителна информация на метод (*формални аргументи*)- част от заглавието на метода
 - Подават се при изпълнението на метод използващ аргументи (*реални аргументи*)- реалните стойности, предавани на метода при изпълнението му

GradeBook.java

```
1 // Fig. 3.4: GradeBook.java
2 // Class declaration with a method that has a parameter.
3
4 public class GradeBook
5 {
6     // display a welcome message to the GradeBook user
7     public void displayMessage( String courseName )
8     {
9         System.out.printf( "welcome to the grade book for\n%s!\n",
10             courseName );
11     } // end method displayMessage
12
13 } // end class GradeBook
```

Формален аргумент за
displayMessage()

Изпълнение на метод *printf()* като
courseName в случая е реален аргумент
за *printf()*

3.4 Деклариране на метод, използващ аргумент

- Методи на обекти от `class Scanner`
 - `nextLine()` прочита следващия ред от Стандартен вход като `String`
 - `next()` прочита следващата дума (разделител е празен символ) от Стандартен вход като `String`
 - `nextInt()` прочита следващата дума (разделител е празен символ) от Стандартен вход като цяло число
 - `nextDouble()` прочита следващата дума (разделител е празен символ) от Стандартен вход като число с плаваща запетая с двойна точност

GradeBookTest.java

```
1 // Fig. 3.5: GradeBookTest.java
2 // Create GradeBook object and pass a String to
3 // its displayMessage method.
4 import java.util.Scanner; // program uses Scanner
5
6 public class GradeBookTest
7 {
8     // main method begins program execution
9     public static void main( String args[] )
10    {
11        // create Scanner to obtain input from command window
12        Scanner input = new Scanner( System.in );
13
14        // create a GradeBook object and assign it to myGradeBook
15        GradeBook myGradeBook = new GradeBook();
16
17        // prompt for and input course name
18        System.out.println( "Please enter the course name" );
19        String nameOfCourse = input.nextLine(); // read a line of input
20        System.out.println(); // outputs a blank line
21
22        // call myGradeBook's displayMessage method
23        // and pass nameOfCourse as an argument
24        myGradeBook.displayMessage( nameOfCourse );
25    } // end main
26
27 } // end class GradeBookTest
```

Изпълнява метод `nextLine` да прочете целия въведен ред

Изпълнява метод `displayMessage` с един аргумент

```
Please enter the course name:
CS101 Introduction to Java Programming
```

```
welcome to the grade book for
CS101 Introduction to Java Programming!
```

Software Engineering факти 3.1

- Обикновено, обекти се създават с ключовата дума **new**. Константите (наричани още- *литерали*) от тип *String* се представят в **двойни кавички** като например "*hello*". *String* константите референции към *String* обекти, които неявно се създават от компилатора на *Java*.

Обичайна грешка при програмиране 3.2

- Грешка при компилация е, когато в тялото на метода има **променлива със същото име, както някой от аргументите на метода в неговата дефиниция.**

Още за аргументи на методи ...

- Списъкът с аргументи на един метод задава неговия “**подпис**” (method signature). Характеризира се с
 - Брой на аргументите
 - Поредност на типовете на аргументите
- Два метода са различни, ако имат
 - Различно име
 - Еднакво име, но различен “подпис”

Например, методите

```
public void    test()    {}  
public    int  test()    {}
```

са **еднакви** (имат еднакво име и еднакъв “подпис”), а методите

```
public void testA(String name) {}  
public int  testA()    {}
```

са **различни**- имат еднакво име, **но** имат различен “подпис”

Обичайна грешка при програмиране 3.2

- Грешка при компилация е, когато в един клас има **два е повече еднакви метода** т.е. в тялото на всеки клас трябва **да има само различни методи.**

Обичайна грешка при програмиране 3.3

- Грешка при компилация се издава винаги когато броят на реалните аргументи е различен от броя на формалните аргументи или реалните аргументи не съответстват по тип и поредност на списъка от формални аргументи, с който е дефиниран метода.

UML Class диаграма за class GradeBook

- **UML class diagram за class GradeBook**
 - Графично представяне за данните и методите на класа
 - В случая class GradeBook има само един public метод (означен с +) който е изобразен с името и списъка с аргументи



Fig. 3.6 | UML class диаграма указваща, че class GradeBook има един `displayMessage` метод с аргумент `courseName` от тип `String`.

Бележки върху *import* декларациите

- *java.lang* се **импортира неявно** във всеки клас на Java
- **Библиотека по подразбиране** (default package)
 - Съдържа всички *class* файлове в текущата директория
 - Тя **също се импортира неявно** в source code на всеки Java файл намиращ се в текущата директория

Например, ако файловете *Circle.java* и *DrawTest.java* са в една и съща директория, то *class DrawTest* може да използва *class Circle* без да има нужда от *import* декларация за *class Circle*, понеже *DrawTest.java* се намира в същата директория с *Circle.java*. Същото се отнася за *class GradeBook* и *class GradeBookTest*

- *import* замества необходимостта от използване на името на съответната библиотека пред името на класа- **напълно определено име на клас**
- Например, **ако не използвате** пред дефиницията на даден клас *import java.util.Scanner;*
то вътре в дефиницията на клас трябва **да пишете всеки път**
java.util.Scanner // пример за напълно определено име на клас
когато искате за използвате клас *Scanner*

Software Engineering факт 3.2

- Java компилаторът не изисква *import* декларации в Java source файла, ако навсякъде се използва *напълно определеното име* за класа (пред името на класа стои името на библиотеката, към която принадлежи класа).
- Все пак **по-удобно е** да се избегне използването *напълно определеното име* за класа **като се напишат** съответните *import декларации* преди дефиницията на класа.

3.5 Данни на клас, *set* методи и *get* методи

- Локални променливи
 - Декларирани в тялото на метода
 - Включват аргументите на метода
 - Достъпни са единствено в тялото на метода
- Данните, *декларирани в тялото на класа и извън методите му*
 - Наричат се *данни, променливи, атрибути* на клас
 - Достъпни са във всеки един от методите на класа
 - Декларират се наедно в началото на тялото на класа (*преди всички методи*), а не разхвърляни между методите на класа
 - Всеки обект от класа получава отделно копие (инстанция) от данните на класа (в случай, че съответната данната не е обща за всички обекти)

Модификатори за достъп

public и *private*

- ***private*** ключова дума
 - Означава данна или метод е **достъпна единствено в тялото на своя клас на дефиниция**
 - **Винаги декларирайте клас данните като *private***
 - реализира **data hiding** (скриване на информация) и гарантира **контролиран достъп** до данните и методите на класа, респ. *правилно реализиране на поведението на обектите от този клас-какво би се получило, ако данните на клас или обект са свободни за произволна промяна? Да си спомним, че всички методи на клас са зависими от данните на класа.*
- ***public*** ключова дума
 - Означава данна или метод, която е **общо достъпна за всички обекти и класове в изпълняваната програма**
 - **В рамките на този курс НИКОГА не декларирайте клас данни *public***
 - Винаги декларирайте ***set*** и ***get*** методите като ***public***
 - Класовете също трябва да са ***public***

Software Engineering факт 3.3

- Пишете съответния модификатор за достъп до всеки метод и клас данна.
- По **правило**, **клас данните** винаги трябва да са *private*, а методите обикновено *public*.

Return type на метод

- - тип на *данните връщани като резултат от работата на метода*
 - Декларира се в заглавието на метода
 - Използва командата *return*, следвана от променлива или константа от същия return type
 - Return type е *void*, когато методът няма за предназначение на връща данни

```
public void insertMoney(int amount)
{
    balance = balance + amount;
} // no return type
```

```
public int insertMoney(int amount)
{
    balance = balance + amount;
    return balance;
} // with return type
```


3.5 *set* методи

- Методи за промяна(**mutator methods**)
 - Водят до промяна в стойността на клас данна- респ. статуса на обекта
 - Служат да зададат нова текущата стойност на една или повече клас данни от други обекти, на които е позволено такова действие
 - **Задължително се проверява валидността на новата стойност**
 - **Регулират кой и как може да променя данните на класа**
- Наричат се още ***set* методи**
 - Задължително името съдържа префикс ***set***, следвано от **името на клас данна**, за чиято стойност ще се променя с метода **като първата буква в името на клас данната се сменя с главна**
 - По правило **клас данните трябва да са недостъпни (***private***)** за други обекти
 - Клас данна **трябва да се променя единствено**, чрез съответен ***set* метод**

Пример:

```
private String courseName;           // клас данна
public void setCourseName(String newCourse){ // set метод за courseName
    if (newCourse != null) // validate!
        courseName = newCourse;
}
```

3.5 *set* методи (Mutator methods)

Set метод за клас данната

```
private int price;
```

visibility modifier

return type

method name

parameter

```
public void setPrice(int newPrice)
{
    if ( newPrice >= 0) // validation
        price = newPrice;
}
```

field being mutated

assignment statement

3.5 *get* методи

- Методи за достъп (**accessor methods**)
 - Не водят до промяна в стойността на клас данна- респ. Статуса на обекта
 - Служат да дадат информация за текущата стойност на една или повече клас данни, за които другите обекти е позволено да узнаят
 - **Регулират достъпа до данните на класа**
 - Да предоставят в подходящ формат стойност на една или повече клас данни
- Наричат се още ***get* методи**
 - Задължително името съдържа префикс ***get***, следвано от **името на клас данна**, за чиято стойност ще се дава информация с метода **като първата буква в името на клас данната се сменя с главна за други обекти**
 - Клас данна се прави достъпна чрез дефиниране на съответен ***get*** метод

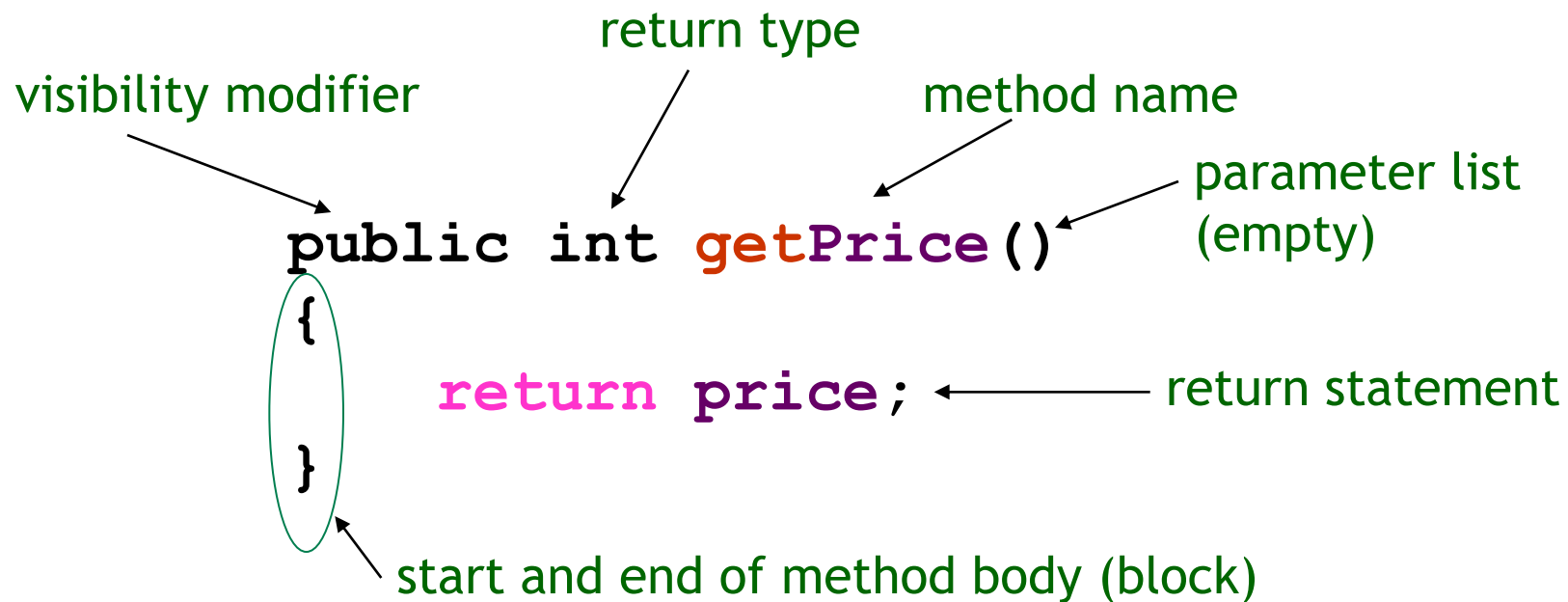
Пример:

```
private String courseName;           // клас данна
public String getCourseName() { // get метод за courseName
    return courseName;
}
```

3.5 *get* методи (Accessor methods)

Get метод за клас данната

```
private int price;
```



```
1 // Fig. 3.7: GradeBook.java
2 // GradeBook class that contains a courseName instance variable
3 // and methods to set and get its value.
```

```
4
5 public class GradeBook
```

Клас данна courseName

GradeBook.java

```
6 {
```

```
7     private String courseName; // course name for this GradeBook
```

```
8
9     // method to set the course name
```

```
10    public void setCourseName( String name )
```

set метод за courseName

```
11    {
```

```
12        courseName = name; // store the course name
```

```
13    } // end method setCourseName
```

```
14
15    // method to retrieve the course name
```

```
16    public String getCourseName()
```

get метод за courseName

```
17    {
```

```
18        return courseName;
```

```
19    } // end method getCourseName
```

```
20
21    // display a welcome message to the GradeBook user
```

```
22    public void displayMessage()
```

```
23    {
```

```
24        // this statement calls getCourseName to get the
```

```
25        // name of the course this GradeBook represents
```

```
26        System.out.printf( "welcome to the grade book for\n%s!\n",
```

```
27            getCourseName() );
```

Извикване на get метод

```
28    } // end method displayMessage
```

```
29
30 } // end class GradeBook
```

Правила за добро програмиране 3.1

- По правило данните на класа се декларираат в началото на тялото на класа преди методите.
- **Недопустимо е** декларацията на данните да е разхвърляна между декларациите на методите.

Правила за добро програмиране 3.2

- **Оставяйте по един празен ред между дефинициите на методите за придаване на по-добра читаемост на програмите.**

class GradeBookTest демонстрира поведението на обекти от class GradeBook

- Създава обект *myGradeBook* от *class GradeBook*
- Демонстрира поведението на така получения обект, чрез изпълнение на методите на този обект, достъпни за *class GradeBookTest*
 - Разпечатва текущата стойност на клас данната *courseName* на обекта *myGradeBook*, чрез *get* метода
 - Референтни данни са *null* по подразбиране
 - Прочита нова стойност за *courseName* от Стандартен вход и присвоява на *courseName* чрез *set* метода

GradeBookTest.java

(1 of 2)

```
1 // Fig. 3.8: GradeBookTest.java
2 // Create and manipulate a GradeBook object.
3 import java.util.Scanner; // program uses Scanner
4
5 public class GradeBookTest
6 {
7     // main method begins program execution
8     public static void main( String args[] )
9     {
10         // create Scanner to obtain input from command window
11         Scanner input = new Scanner( System.in );
12
13         // create a GradeBook object and assign it to myGradeBook
14         GradeBook myGradeBook = new GradeBook();
15
16         // display initial value of courseName
17         System.out.printf( "Initial course name is: %s\n\n",
18             myGradeBook.getCourseName() );
19     }
```

Изпълнява *get* метода за
courseName

```
20 // prompt for and read course name
21 system.out.println( "Please enter the course name:" );
22 String theName = input.nextLine(); // read a line of text
23 myGradeBook.setCourseName( theName ); // set t
24 system.out.println(); // outputs a blank line
25
26 // display welcome message after specifying course name
27 myGradeBook.displayMessage();
28 } // end main
29
30 } // end class GradeBookTest
```

Изпълнява *set* метода за
courseName

Изпълнява *displayMessage*

```
Initial course name is: null
Please enter the course name:
CS101 Introduction to Java Programming

welcome to the grade book for
CS101 Introduction to Java Programming!
```

GradeBookTest.java
(2 of 2)

UML Class диаграма за class GradeBook

- **Данни на класа**
 - **Описани в средната част**
 - **Името на данната следвано от две точки и типа на данната**
- **Return type на метод**
 - **Указва се след името на метода с две точки, следвано от типа на данните, връщани от метода**

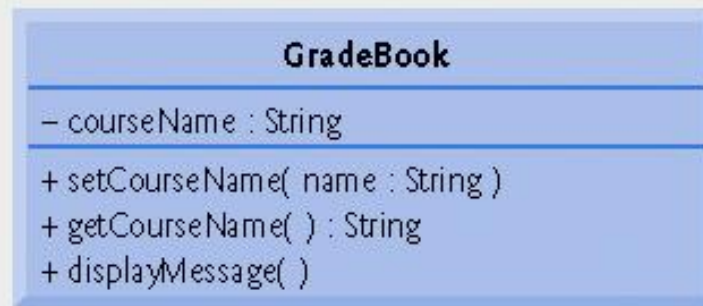


Fig. 3.9 | UML class диаграма за class GradeBook.

Сравнение между примитивни и референтни типове данни

- Типове данни в Java- *всяка променлива име определен тип*
 - примитивни
 - *boolean, byte, char, short, int, long, float, double*
 - Инициализират се с конкретна стойност **по подразбиране**
boolean → *false*
 - Всички останали взимат числено представяне на **нула**
 - референтни (или *непримитивни* типове)
 - Референции към обекти на класове
 - Получават стойност *null* по подразбиране (референция към "никъде")
 - не могат да се използват преди да се инициализират на обект от даден клас
 - Позволяват извикване на методи на обект
- Пример. *myGradeBook.setCourseName(theName);*

Software Engineering факт 3.4

- Типът на променливата, използван при нейната декларация (т.е., `int`, `double` или `GradeBook`) указва дали променливата е от примитивен или референтен тип. Ако типът на променливата не е някой от изброените примитивни типове- то тя е от референтен тип. Например,

`Account account1 ;`

Показва, че `account1` е референция към обект от клас `Account`.

3.7 Инициализиране на обекти с конструктори

- Конструктори

- Специализирани методи за създаване на обект от клас
- Дефинират се **с абсолютно същото име като класа**, на когото принадлежат
- **Нямат return type** и **не може да използват** командата ***return***
- Най- често се дефинират като ***public***
- Java **изисква използване на конструктор** за всеки клас
- **Извикват** се когато ключовата дума ***new*** е последвана от име на клас и двойка кръгли скобки(отваряща и затваряща)

3.7 Инициализиране на обекти с конструктори

- Видове конструктори

- **Конструктор по подразбиране** (default constructor)- *списъкът с аргументи е празен* и се създава обект с предварително зададен начален статус

```
public GradeBook() { ...}
```

- **Конструктор за общо ползване** (general purpose constructor) - *списъкът с аргументи съответства по брой и тип на данните на класа* и служи за създаване на обект с конкретно зададен начален статус

```
public GradeBook(String name) { ...}
```

- **Конструктор за копиране** (copy constructor)- *списъкът с аргументи съдържа само един елемент*, референция към обект от същия клас и служи за създаване на нов обект със същия начален статус както реферирания обект

```
public GradeBook(GradeBook someGradeBook) { ...}
```


GradeBook.java

(1 of 2)

```
1 // Fig. 3.10: GradeBook.java
2 // GradeBook class with a constructor to initialize the course name.
3
4 public class GradeBook
5 {
6     private String courseName; // course name for this GradeBook
7
8     // constructor initializes courseName with String supplied as argument
9     public GradeBook( String name )
10    {
11        setCourseName(name); // initializes courseName using the set method
12    } // end constructor
13
14    // method to set the course name
15    public void setCourseName( String name )
16    { if (name != null) // validation!!!!
17        courseName = name; // store the course name
18    } // end method setCourseName
19
20    // method to retrieve the course name
21    public String getCourseName()
22    {
23        return courseName;
24    } // end method getCourseName
```

Конструктор за **общо ползване**

Инициализира **ВСИЧКИ** данни
на инстанцията посредством **set**
методите им

set методите задължително
ВАЛИДИРАТ и **присвояват само**
валидни стойности на данните
на инстанцията

GradeBook.java

(2 of 2)

```
25
26 // display a welcome message to the GradeBook user
27 public void displayMessage()
28 {
29     // this statement calls getCourseName to get the
30     // name of the course this GradeBook represents
31     System.out.printf( "welcome to the grade book for\n%s!\n",
32         getCourseName() );
33 } // end method displayMessage
34
35 } // end class GradeBook
```

GradeBookTest.java

```
1 // Fig. 3.11: GradeBookTest.java
2 // GradeBook constructor used to specify the course name at the
3 // time each GradeBook object is created.
4
5 public class GradeBookTest
6 {
7     // main method begins program execution
8     public static void main( String args[] )
9     {
10         // create GradeBook object
11         GradeBook gradeBook1 = new GradeBook(
12             "CS101 Introduction to Java Programming" );
13         GradeBook gradeBook2 = new GradeBook(
14             "CS102 Data Structures in Java" );
15
16         // display initial value of courseName for each GradeBook
17         System.out.printf( "gradeBook1 course name is: %s\n",
18             gradeBook1.getCourseName() );
19         System.out.printf( "gradeBook2 course name is: %s\n",
20             gradeBook2.getCourseName() );
21     } // end main
22
23 } // end class GradeBookTest
```

Извиква конструктор за създаване на първия grade book обект

Създаване на втори grade book обект

```
gradeBook1 course name is: CS101 Introduction to Java Programming
gradeBook2 course name is: CS102 Data Structures in Java
```

Правила за добро програмиране 3.2

- Конструкторите се пишат веднага след декларациите на клас данните, **преди всички останали методи** на класа.
- **След тях** се пишат `set` и `get` методите за съответните клас данни.
- После се пишат всички останали методи на класа

Правила за добро програмиране 3.2

- Дефинирането на **трите вида конструктори** във всеки клас е задължително условие за правилното логически издържано създаване на обекти в Java.
- **Абсолютно задължително е** всеки клас да има конструктор за общо ползване, когато класът има клас данни.

Включване на конструктор UML Class диаграмата на `class GradeBookTest`

- **UML клас диаграма**
 - Конструкторите се описват долната част на класа, но над всеки от методите на класа
 - Вмъкнете “<<constructor>>” пред името на конструктора
 - По правило, описвайте конструкторите преди всички останали методи в UML диаграмата

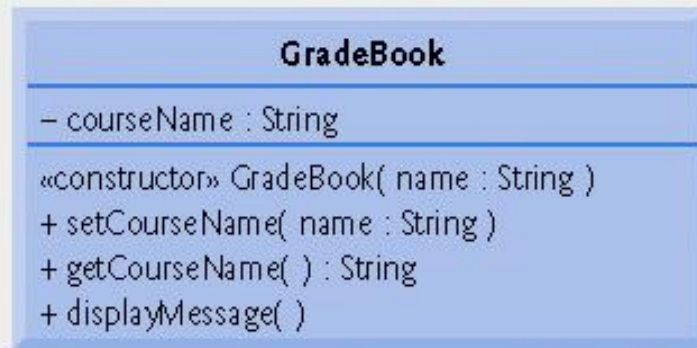


Fig. 3.12 | UML клас диаграма показваща, че клас *GradeBook* има конструктор с аргумент от тип *String*.

3.8 Числа с плаваща запетая и тип данни `double`

- **Числа с плаваща запетая**

- *`float`*

- константите се означават `0.5f`, `-2.0f`

- *`double`*

- Използва двойно повече памет от `float` и съответно дава по- голяма точност (броят цифри след десетичната точка) при пресмятане от `float`

- константите се означават **`0.5d`**, **`-2.0d`**
 - Константите по подразбиране са `double`!

Обичайна грешка при програмиране 3.3

Грешка при компилация се издава винаги когато променлива от тип с по-малка точност се присвоява **стойност от тип с по-голяма точност**

Пример:

float f = 2.0; // грешка, 2.0 е double по подразбиране

float f = 2.0f; // правилно, 2.0 е дадена като float константа

Числа с плаваща запетая и тип данни `double`

- ***float***

- **единична-точност за floating-point numbers (32 бита)**
- **7 значещи цифри след десетичната точка**

- ***double***

- **двойна-точност за floating-point numbers (64 бита)**
- **15 значещи цифри след десетичната точка**

Account.java

```
1 // Fig. 3.13: Account.java
2 // Account class with a constructor to
3 // initialize instance variable balance.
4
5 public class Account
6 {
7     private double balance; // instance variable that stores the balance
8
9     // constructor
10    public Account( double initialBalance )
11    {
12        // validate that initialBalance is greater than 0.0;
13        // if it is not, balance is initialized to the default value 0.0
14        if ( initialBalance > 0.0 )
15            balance = initialBalance;
16    } // end Account constructor
17
18    // credit (add) an amount to the account
19    public void credit( double amount )
20    {
21        balance = balance + amount; // add amount to balance
22    } // end method credit
23
24    // return the account balance
25    public double getBalance()
26    {
27        return balance; // gives the value of balance to the calling method
28    } // end method getBalance
29
30 } // end class Account
```



double variable balance

class AccountTest **тества** class Account

- **Форматен спесификатор %f**
 - Използва се за данни от тип плаваща запетая
 - **Числото след знака** за процент и **f** задава цифрите след десетичната точка, които ще се вземат при преобразуването на числото в текст

Пример:

%0.5f - само **5** цифри след десетичната запетая ще се вземат

%8.2f – числото ще се представи фиксирано в **8** символни позици с **2** цифри след десетичната запетая

AccountTest.java

(1 of 3)

```
1 // Fig. 3.14: AccountTest.java
2 // Create and manipulate an Account object.
3 import java.util.Scanner;
4
5 public class AccountTest
6 {
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10         Account account1 = new Account( 50.00 ); // create Account object
11         Account account2 = new Account( -7.53 ); // create Account object
12
13         // display initial balance of each object
14         System.out.printf( "account1 balance: $%.2f\n",
15             account1.getBalance() );
16         System.out.printf( "account2 balance: $%.2f\n\n",
17             account2.getBalance() );
18     }
```

```
19 // create Scanner to obtain input from command window
20 Scanner input = new Scanner( System.in );
21 double depositAmount; // deposit amount read from user
22
23 System.out.print( "Enter deposit amount for account1: " ); // prompt
24 depositAmount = input.nextDouble(); // obtain user input
25 System.out.printf( "\nadding %.2f to account1 balance\n",
26     depositAmount );
27 account1.credit( depositAmount ); // add to account1 balance
28
29 // display balances
30 System.out.printf( "account1 balance: $%.2f\n",
31     account1.getBalance() );
32 System.out.printf( "account2 balance: $%.2f\n\n",
33     account2.getBalance() );
34
35 System.out.print( "Enter deposit amount for account2: " ); // prompt
36 depositAmount = input.nextDouble(); // obtain user input
37 System.out.printf( "\nadding %.2f to account2 balance\n\n",
38     depositAmount );
39 account2.credit( depositAmount ); // add to account2 balance
40
```

Въвеждане на double стойност

AccountTest.java
(2 of 3)

Въвеждане на double
стойност

AccountTest.java

```
41 // display balances
42 System.out.printf( "account1 balance: $%.2f\n",
43     account1.getBalance() );
44 System.out.printf( "account2 balance: $%.2f\n",
45     account2.getBalance() );
46 } // end main
47
48 } // end class AccountTest
```

Извеждане на double стойност

AccountTest.java
(3 of 3)

AccountTest.java

```
account1 balance: $50.00
account2 balance: $0.00
```

```
Enter deposit amount for account1: 25.53
```

```
adding 25.53 to account1 balance
```

```
account1 balance: $75.53
account2 balance: $0.00
```

```
Enter deposit amount for account2: 123.45
```

```
adding 123.45 to account2 balance
```

```
account1 balance: $75.53
account2 balance: $123.45
```

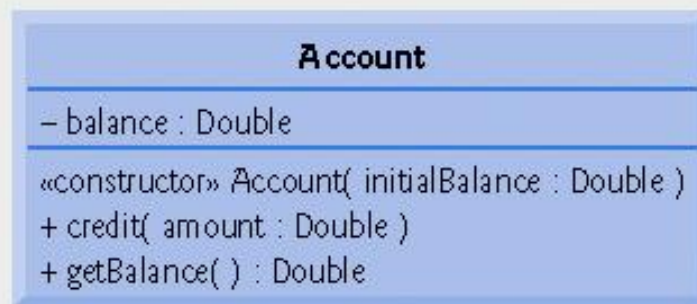


Fig. 3.15 | UML class diagram показваща че class **Account** има **private balance** данна от тип **Double**, а също конструктор (с аргумент от тип **Double**) и два **public** метода—**credit** (с аргумент от тип **Double**) и **getBalance** (връща тип **Double**).

3.6 Обобщение

- Всяка декларация на клас изисква ключовата дума ***class*** непосредствено следвана от името на класа.
- Декларация на метод започваща с ключовата дума ***public*** указва, че е “общо достъпен” и съответно този метод може да се извиква от други класове дефинирани извън дефиницията на този клас.
- Ключовата дума ***void*** указва, че дадения метод ще изпълни действие, но няма да върне информация след приключване на изпълнението си.
- По правило, имената на методи започват с малка буква и следват стила на “камилата” – всяка следваща дума в името на метода е с главна първа буква.
- Празни кръгли скобки след името на метода, указват, че методът не изисква параметри, за да изпълни заданието си.
- Тялото на всеки метод е ограничено с двойка отваряща и затваряща фигурна скобка (***{*** и ***}***).
- Аргументите и променливите деклариран в тялото на метода се наричат локални променливи.
- Локалните променливи на метод са валидни само в тялото на метода, където са деклариран.

3.6 Обобщение

- Обектите на всеки клас се инициализират от специален метод наречен конструктор
- Има три вида конструктори- по подразбиране, за общо ползване и конструктор за копиране. Най- често тези конструктори са общо достъпни и използват *public* модификатор за достъп.
- Името на конструктора съвпада с името на класа, чиито обекти конструира.
- Конструктор не може да има тип на връщане на данни, нито да използва командата `return`.
- Изразът за създаване на обект от клас започва с ключовата дума *new* следвана от извикване на някой от конструкторите на класа.
- За изпълнение на метод на обект, този обект трябва да е първо създаден. За извикване на метода използваме име на променливата, която реферира този обект, следвана от точка и името на метода и неговите аргументи, заградени в кръгли скоби.
- *Scanner* метода *nextLine()* чете от клавиатурата символи, докато срещне символ за нов ред и тогава връща прочетените символи като текст- *String*.
- *Scanner* метода *next()* чете от клавиатурата символи, докато срещне поредица от празни символи и тогава връща прочетените символи като текст- *String*.

3.6 Обобщение

- Всеки клас има един или повече методи, които манипулират данните на класа. Данните описват текущото състояние на всеки обект от този клас.
- По правило данните на всеки клас са с `private` модификатор за достъп. Това правило реализира концепцията *data hiding* в ООП
- Методи, които дават информация за текущата стойност на клас данна се наричат методи за достъп- *accessor* методи. По правило тяхното име започва с префикса *get*.
- Методи, които служат за промяна на текущата стойност на клас данна се наричат методи за промяна- *mutator* методи. По правило тяхното име започва с префикса *set*.
- По правило *get* и *set* методите на всеки клас са с `public` модификатор за достъп
- Различаваме примитивни и референтни типове данни. Примитивните данни се инициализират при декларацията им на определени стойности. Референтните променливи се инициализират на неопределена стойност, означена с ключовата дума *null*.
- Променлива от референтен тип не може да се използва, ако предварително не е инициализирана на обект от същия референтен тип както този и апроменливата.

3.7 Задачи

1. Добавете в *class Account* (Fig. 3.13) метод с име *debit()*, който позволява да се теглят пари от обекти на *class Account*. Нека метода проверява дали сумата за дебит не надвишава баланса на съответния обект *Account*. Ако сумата за дебит надвишава този баланс, то балансът трябва да се остави без промяна и метода да изведе съобщение в диалогов прозорец "*Debit amount exceeded account balance.*" Направете необходимите промени в *class AccountTest* (Fig. 3.14) за да тествате метода *debit()*.

3.7 Задачи

2. Променете *class GradeBook* (Fig. 3.10) така че:

- Да има втора клас данна от тип ***String*** за представяне на името на преподавателя на курса
- Напишете ***set*** метод за промяна на името на инструктора , а също и съответен ***get*** метод за намиране на текущата стойност на тази данна.
- Променете конструктора за общо ползване, така че да инициализира двете клас данни- ***courseName*** и новата данна за името на преподавателя.
- Променете метода ***displayMessage()*** така че първо да извежда текст за поздрав (***welcome message***) и името на курса, а след това да извежда и текста "***This course is presented by:*** " следвано от името на преподавателя.

Използвайте така променения клас в тест приложение, което да демонстрира новите възможности на този клас

3.7 Задачи

3. Напишете *class Invoice*, който може да се използва от магазин за хадуер при издаване на фактури за стока, продадена в магазина.

Един обект от този клас (фактура) трябва да има 4 клас данни –

- код на компонента (part number) от тип *String*
- описание на компонентата (part description) от тип *String*
- Бройки на закупената компонента (quantity) от тип *int*
- Цена на компонентата (price) от тип *double*.

Вашият клас трябва да има конструктор за общо ползване, който да инициализира 4- те клас данни.

Напишете *set* и *get* метод за всяка ат клас данните, а за всяка от клас данните приложете концепцията за скриване на информация (*data hiding*)

В допълнение, напишете и метод *getInvoiceAmount()* за пресмятане на общата сума по фактурата (т.е. Умножава бройките закупени от компонентата по нейната цена), итогава връща тази сума като *double* тип данна. Ако закупеният брой от компонентата е отрицателен, сумата на фактурата да се зададе нула. Ако цената на компонентата отрицателна сумата на фактурата да се зададе нула.

Напишете отделен клас за тестване на приложението и именувайте този клас като *InvoiceTest*. Напишете пълен набор от команди, които да демонстрират по- горе описаните задания като правилно изпълнени.

3.7 Задачи

4. Напишете *class Date* който да има три клас данни:

- месец (*month*) от тип *int*
 - ден (*day*) от тип *int*
 - година (*year*) от тип *int*
- *class Date* да има конструктор за общо ползване.
 - *class Date* да има *set* и *get* метод за всяка клас данни- не забравяйте проверките за валидност в съответния метод.
 - Добавете метод *displayDate()* за извеждане на *month*, *day* и *year*, разделени с (/).
 - Напишете тест приложение наречено *DateTest* за пълна демонстрация на поведението на *class Date*

3.7 Задачи

5. Напишете *set* метод за данната *balance* в *class Account* (*Fig. 3.13*). Използвайте този метод в конструктора на *class Account* за инициализиране на данната *balance* вместо съществуващата там *if* команда. Тествайте с *class AccountTest* така написания *class Account* с примерни данни, които се въвеждат с диалогов прозорец. Извеждайте получените резултати в диалогов прозорец.