

Лекция 6.1

Особености при използването на методи в Java

Основни теми

- Особености в използването на методи в Java.
- Статични методи и клас данни.
- Използване на библиотечни статични методи в *class Math*
- Предаване на данни между методи- изпълнение на методи и връщане на данни
- Обсег на валидност на декларации на данни
- Приложения- Генериране на случайни стойности
- Допълнително дефиниране на методи (*overloading*)
- Задачи

- 6.1 Въведение
 - 6.2 Програмни модули в Java
 - 6.3 *static* методи, *static* клас данни и *class Math*
 - 6.4 Деклариране на методи с много аргументи
 - 6.5 Обобщение на декларирането и използването на методи
 - 6.6 Машинно представяне на изпълнението на методи
 - 6.7 Преобразуване на типове данни
 - 6.8 Генериране на случайни числа- приложения
 - 6.9 Обсег на валидност за декларация на данни
 - 6.10 Допълнителни дефиниции на методи
- Задачи

Литература:

Java How to Program, Sixth Edition, глава 6

6.1 Въведение

- Приложните програми на практика са много по- сложни от разглежданите тук примери
- За по- лесна **разработка** и **поддържане** те се конструират от по- малки части, наречени **модули**
- Програмните модули се обособяват с прилагане на техника, наречена “*Разделяй и Владей*”
 - Разбиват една голяма задача на по- малки части (**modules**)
 - На най- ниско ниво тази техника се свежда до подходящо **дефиниране на клас методи**
- Примери: мутатор и аксесор методи
- В тази лекция ще научим как *static* методи на даден клас могат да се изпълняват без да е необходим обект от този клас (вече видяхме *Math.pow()* , *System.out*)

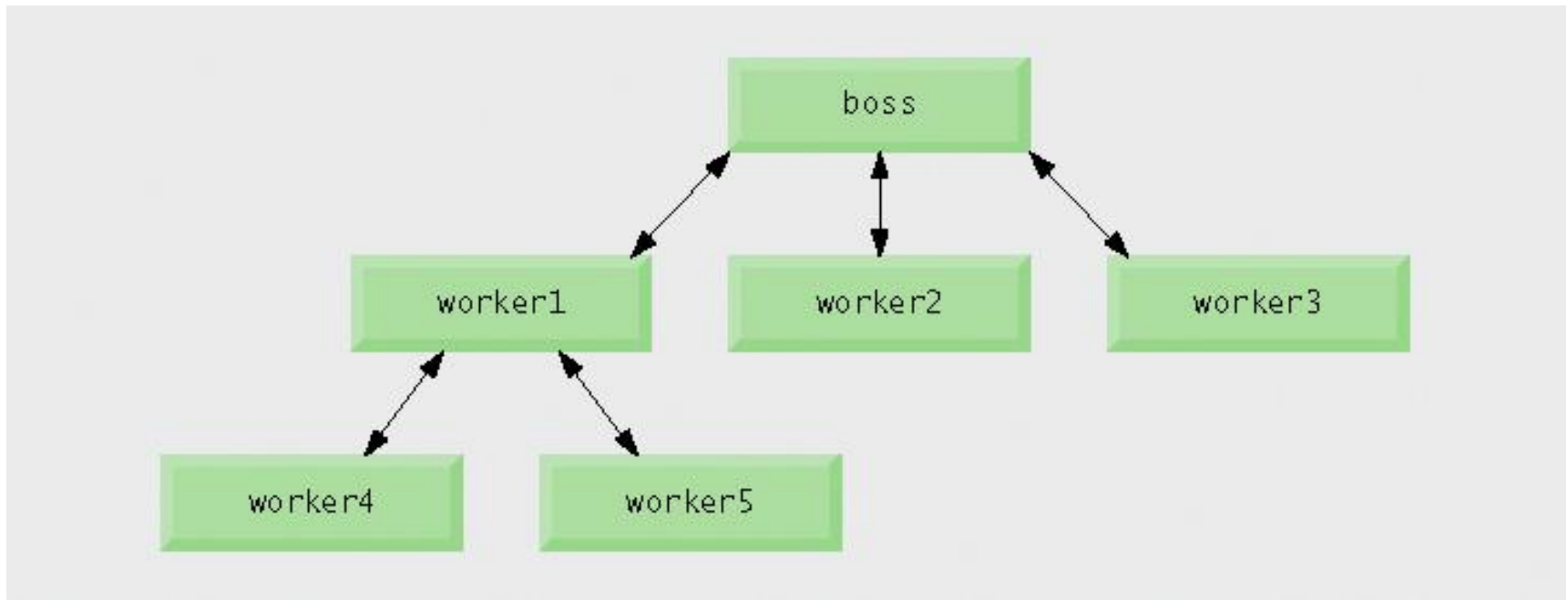


Fig. 6.1 | Hierarchical boss-method/worker-method relationship.

6.1 Въведение

- Тук ще научим как се дефинират аргументи на метод. Ще поясним как Java следи кой метод текущо се изпълнява, как се **представят локалните данни в машинната памет** и как един метод знае **къде да предаде управлението на логиката** след приключване на изпълнението си.
- За илюстрация ще разгледаме способите за **генерация на случайни числа**. Допълнително ще научим как се декларират данни, които не трябва да се менят в процеса на изпълнение на програмата- **константи**.
- Често се налага метод с едно и също име да се извиква с различен набор от аргументи(например, **конструктор** по подразбиране и конструктор за общо ползване) Тази техника е известна като **overloading** (допълнително дефиниране на метод) и се използва за изпълнение на сходни дейности , но с различен брой или тип данни на аргументите.

6.2 Програмни модули в Java

- Java поддържа **3 типа модули**- *методи, класове и библиотеки (packages)*
- Java програми комбинират потребителски дефинирани класове и методи с библиотечни (предефинирани) класове и методи от Java Application Programming Interface (API)
 - Известна е и като Java клас библиотеки
 - Съдържа предефинирани класове и методи
 - Логически свързани класове са обособени в пакети библиотеки (packages)
 - Пример: математически методи, обработка на текст и символи, вход и изход на данни, обработка на файлове и пр.

Software Engineering факти 6.1

“Не преокривайте колелото”. Когато е възможно използвайте многократно Java API класове и методи. Това съкращава времето за разработка на програмата и намалява риска от програмни грешки.

6.2 Програмни модули в Java ...

- **Методи в Java**

- Наричат се **функции** или **процедури** в термините на други програмни езици
- **Структурират** програмите в модули, чрез разделяне на задачите в по- малки и лесно изпълними подзадачи (действия)
- Позволяват да се приложи техниката за “*Разделяй и Владей*”
- Обхващат **блок от команди**, които се използват многократно в изпълнението на програмата (*ре-факторизиране!*)
- **Избягва се повторно писане** на едни и същи групи от команди

Software Engineering факти 6.2

За осъществяване на многократно използване на софтуер **всеки метод** трябва да се свежда до *изпълнение на една добре дефинирана задача и името на метода да съответства на тази задача*. Това оприначя за по- лесното писане, тестване, поддръжка и редактиране на програмите.

Обичайна грешка при програмиране 6.1

Един кратък метод за изпълнение на задача е по-лесно да се тества от един по-дълъг метод, който е проектиран да изпълнява множество от задачи.

Пишете методи не по-дълги от един екран.

Software Engineering факти 6.3

Ако имате **проблем да дадете смислено име** за някой метод, то най често този метод е проектиран да изпълнява много разнородни задачи и това е сигнал да се промени модела на този метод. За целта **се препоръчва** такъв метод да **се разбие на няколко по- малки метода**.

6.3 *static* МЕТОДИ, *static* КЛАС ДАННИ И *class Math*

- **static** метод (метод общ за всички обекти от даден клас)
 - Принадлежи на класа като цяло, вместо на всеки обект поотделно
 - **static метод не зависи от статуса** (текущото състояние, стойностите на клас данните) на отделен обект на класа
 - Извикването на **static** метод става по следната схема:
ClassName.methodName(arguments)
 - Пример: всички методи на **class Math** са **static**
 - `Math.sqrt(900.0)` връща `double` от $\sqrt{900.00}$
 - `Math.pow(2.0, 2.5)`
 - `Math.sin(3.1415)`

Метод	Описание	Пример
<code>abs(x)</code>	Абсолютна стойност на x	<code>abs(23.7)</code> е 23.7 <code>abs(0.0)</code> е 0.0 <code>abs(-23.7)</code> е 23.7
<code>ceil(x)</code>	закръгля x до най- малкото цяло не по-малко от x	<code>ceil(9.2)</code> е 10.0 <code>ceil(-9.8)</code> е -9.0
<code>cos(x)</code>	косинус x (x в радиани)	<code>cos(0.0)</code> е 1.0
<code>exp(x)</code>	Ескпонента на x	<code>exp(1.0)</code> е 2.71828 <code>exp(2.0)</code> е 7.38906
<code>floor(x)</code>	закръгля x до най- голямото цяло не по-голямо от x	<code>floor(9.2)</code> е 9.0 <code>floor(-9.8)</code> е -10.0
<code>log(x)</code>	Естествен логаритъм на x (основа e)	<code>log(Math.E)</code> е 1.0 <code>log(Math.E * Math.E)</code> е 2.0
<code>max(x, y)</code>	По- голямото между x и y	<code>max(2.3, 12.7)</code> е 12.7 <code>max(-2.3, -12.7)</code> е -2.3
<code>min(x, y)</code>	По- малкото между x и y	<code>min(2.3, 12.7)</code> е 2.3 <code>min(-2.3, -12.7)</code> е -12.7
<code>pow(x, y)</code>	x на степен y	<code>pow(2.0, 7.0)</code> е 128.0 <code>pow(9.0, 0.5)</code> е 3.0
<code>sin(x)</code>	Синус от x (x е в радиани)	<code>sin(0.0)</code> е 0.0
<code>sqrt(x)</code>	Корен квадратен от x	<code>sqrt(900.0)</code> е 30.0
<code>tan(x)</code>	Тангенс от x (x е в радиани)	<code>tan(0.0)</code> е 0.0

Fig. 6.2 | Math class методи.

6.3 *static* МЕТОДИ, *static* КЛАС ДАННИ И *class Math*

- **Деклариране** на `static` метод

- Използва се ключовата дума `static` пред типа на връщаните данни

- Спомнете си

```
public static void main(String[] args) {}
```

Принадлежи на класа като цяло, вместо на всеки обект поотделно

Software Engineering факти 6.4

class Math принадлежи на **java.lang** пакета, който се имротира неявно, така че няма нужда д асе импортира **class Math** за да се използват неговите методи.

6.3 *static* МЕТОДИ, *static* клас данни и *class Math* ...

- **Константи**
 - Декларират се с ключова дума **final**
 - **Инициализират** се на мястото на декларацията им
 - **Не могат да се променят** след инициализация
 - В общия случай всеки обект може да има своя стойност за дадена константа
- **static клас данни** (клас атрибути, полета пр.)
 - Такива данни са **общи** за всички обекти от даден клас
 - Могат да се **инициализират** след като са декларирани
 - **Могат да се променят** в процеса на изпълнението

Примери:

- **Math.PI** (числото PI) и **Math.E** (числото e) са
- **final static** клас данни на **class Math**
- Следователно, това са **константи общи за всички обекти** от този клас.

6.3 *static* МЕТОДИ, *static* КЛАС ДАННИ И *class Math* ...

- **Метод `main()`**
 - `main()` е деклариран ***static***, за да може да се изпълни без да има нужда от създаване на обект от класа съдържащ метода `main()`
 - Всеки отделен клас може да има `main()` метод
 - JVM извиква `main()` метода принадлежащ на класа, зададен като първи аргумент при извикване на `java` командата
 - На този етап няма създадени обекти от класа, зададен като първи аргумент при извикване на `java` командата
 - JVM изпълнява `main()` метода като задава за негови аргументи следващи името на класа в `java` командата

6.3 *static* методи, *static* клас данни и *class Math* ...

- Пример

- `main()` е деклариран като

- ```
public static void main(String args[])
```

- При изпълнение на командата

- ```
java ClassName argument1 argument2 ...
```

- JVM изпълнява `main()` на `ClassName` със списък от аргументи (от тип *String*)

- ```
argument1, argument2 , ...
```

- По такъв начин изпълнението на `ClassName` може да зависи от определен набор параметри, задавани на командния ред (*ще обясним как в следващата лекция*)

## 6.4 Деклариране на методи с много аргументи

- Повече от един аргумент се декларира в списък от декларации на данни, разделени със запетая.
- Тези данни са част от локалните данни на метода и се наричат още- **формални аргументи**
- При изпълнение на метод на всеки формален аргумент се съпоставя променлива, реферираща стойност или обект съответстваща по тип на формалния аргумент- в този случай данните се наричат още **реални аргументи**

## 6.4 Деклариране на методи с много аргументи

- **Пример:** Fig. 6.3 и Fig. 6.4 използва потребителски дефиниран метод `maximum()` за намиране и връщане на най- голямото от 3 числа (`double`) въведени от потребителя.
- В началото на изпълнението, метода `main()` на `class MaximumFinderTest` (редове 7- 11 от Fig. 6.4) създава обект от `class MaximumFinder` (ред 9) и извиква метода `determineMaximum()` (ред 10) за извеждане на крайните резултати.
- В `class MaximumFinder` (Fig. 6.3), редове 14- 18 от метод `determineMaximum()` известяват потребителя да въведе 3 `double` числа и ги прочита от клавиатурата.
- Ред 21 извиква метод `maximum` (редове 28- 41) за определяне на най- голямото от въведените числа. Когато метод `maximum()` връща резултата на ред 21, програмата присвоява крайния резултат от изпълнението на `maximum()` на локалната променлива `result`.
- Тогава ред 24 извежда крайния резултат.

## Outline

MaximumFinder.java

(1 of 2)

```

1 // Fig. 6.3: MaximumFinder.java
2 // Programmer-declared method maximum.
3 import java.util.Scanner;
4
5 public class MaximumFinder
6 {
7 // obtain three floating-point values and locate the maximum value
8 public void determineMaximum()
9 {
10 // create Scanner for input from command window
11 Scanner input = new Scanner(System.in);
12
13 // obtain user input
14 System.out.print(
15 "Enter three floating-point values separated by spaces: ");
16 double number1 = input.nextDouble(); // read first double
17 double number2 = input.nextDouble(); // read second double
18 double number3 = input.nextDouble(); // read third double
19
20 // determine the maximum value
21 double result = maximum(number1, number2, number3);
22
23 // display maximum value
24 System.out.println("Maximum is: " + result);
25 } // end method determineMaximum
26

```

Съобщава на потребителя  
да въведе три **double**  
числа

Извиква метод **maximum** с три  
**реални аргумента** - по  
**тип и брой** съответстват  
на формалните

Съответства по тип на  
връщаната от  
метода данна

Извежда максималната  
стойност



```
27 // returns the maximum of its three double parameters
28 public double maximum(double x, double y, double z)
29 {
30 double maximumValue = x; // assume x is the largest to start
31
32 // determine whether y is greater than maximumValue
33 if (y > maximumValue)
34 maximumValue = y;
35
36 // determine whether z is greater than maximumValue
37 if (z > maximumValue)
38 maximumValue = z;
39
40 return maximumValue;
41 } // end method maximum
42 } // end class MaximumFinder
```

Декларация на метода  
**maximum** с три  
формални аргумента

MaximumFinder.java

(2 of 2)

Сравнява **y** и **maximumValue**

Сравнява **z** и **maximumValue**

Връща намерената максимална стойност,  
съответства на декларирания тип за  
връщане на данни в заглавието на метода



## Outline

MaximumFinderTest  
.java

```
1 // Fig. 6.4: MaximumFinderTest.java
2 // Application to test class MaximumFinder.
3
4 public class MaximumFinderTest
5 {
6 // application starting point
7 public static void main(String args[])
8 {
9 MaximumFinder maximumFinder = new MaximumFinder();
10 maximumFinder.determineMaximum();
11 } // end main
12 } // end class MaximumFinderTest
```

Създава обект от клас  
**MaximumFinder**

Изпълнява метод  
**determineMaximum**

Enter three floating-point values separated by spaces: 9.35 2.74 5.1  
Maximum is: 9.35

Enter three floating-point values separated by spaces: 5.8 12.45 8.32  
Maximum is: 12.45

Enter three floating-point values separated by spaces: 6.46 4.12 10.54  
Maximum is: 10.54





# Обичайна грешка при програмиране 6.1

---

**Декларирането на аргументи на метод от един и същ тип като**

**`float x, y`**

**вместо**

**`float x, float y`**

**е синтактична грешка- декларирането на типа се изисква за всеки отделен аргумент в списъка с аргументи на метода**

# Software Engineering факти 6.5

---

**Метод с дълъг списък от аргументи** вероятно изпълнява твърде много задачи.

Помислете за разделянето на такъв метод на няколко по-малки методи.

Старайте се **списъка с аргументи да не надхвърля един ред.**

## 6.4 Деклариране на методи с много аргументи...

- Многократно използване на методи (пример, метод **Math.max**)
  - Изразът **Math.max( x, Math.max( y, z ) )** пресмята максимума от **y** и **z**, а след това пресмята максимума от **x** и предишната пресметната стойност
  - Така редове 20 -21 от Fig 6.3 може да се пренапишат като

```
20 // determine the maximum value
21 double result = Math.max(x, Math.max(y, z)) ;
```

## 6.4 Деклариране на методи с много аргументи...

- Събиране на низове (String concatenation)
  - Използването на оператора `+` с два `Strings` ги събира в нов низ (`String`)
  - Използването на оператора `+` с един `String` и стойност от друга стойност е сумарен `String` с текстовото представяне първия низ и `String` представянето на втория операнд на оператора `+`
  - Когато вторият операнд е обект, то се извиква неговия `toString()` метод (ако е дефиниран) за генериране на `String` представянето на този обект
  - Методът `toString()` трябва да е дефиниран като

```
public String toString() {}
```

## Обичайна грешка при програмиране 6.2

---

**Синтактична грешка е да се разбие една `String` константа (literal) на няколко реда в програмата.**

**Когато един `String` не се събира на един ред, разбийте го на части и използвайте събиране на низове, за да представите зададения стринг , чрез неговите части.**

## Обичайна грешка при програмиране 6.3

---

Объркването на  $+$  оператора, използван за събиране на низове и  $+$  оператора, използван за събиране на числа може да доведе до странни резултати.

Java пресмята операндите на оператор отляво надясно. Например, ако целочислената  $y$  е 5, то изразът

`"y + 2 = " + y + 2`

дава низа

`"y + 2 = 52"`, а не

`"y + 2 = 7"`,

понеже първо  $y$  (5) се събира с низа `"y + 2 = "`, а после 2 се събира с низа `"y + 2 = 5"`.

Изразът `"y + 2 = " + (y + 2)` дава желаният резултат `"y + 2 = 7"`.

## 6.5 Обобщение на декларирането и използването на методи

- Има 3 начина за **извикване на метод**:
  - Използване само на името на метода (**директно** извикване)
  - Използване на променлива, реферираща обект и следвана от точка (.) и името на метода, принадлежащ на реферирания метод
  - Използване на име на клас, следвано от име на клас и точка (.) за извикване на `static` метод от този клас

### Важно:

- `static` методите **не могат** да извикват **директно** не-`static` методи от същия клас
- `static` методите могат да извикват директно само други `static` методи от същия клас

## 6.5 Обобщение на декларирането и използването на методи

- Има 3 начина за **връщане на управлението** на след изпълнение на метод :
    - Ако методът не връща данни (деклариран **е от тип *void***), методът прекратява изпълнението си при :
      - Достигане на затваряща фигурна скоба за блока от команди на метода
      - Методът изпълнява командата `return ;`
    - Ако методът връща данни (деклариран е **от тип различен** на ***void***), като краен резултат:
      - Методът изпълнява командата `return expression ;`
- Където ***expression*** се пресмята до тип данни, **съответстващи на типа данни в заглавието на метода** и после **стойността на данните се връща** на викащия метод



## Обичайна грешка при програмиране 6.4

---

**Декларирането на метод извън тялото на декларацията на клас или в тялото на друг метод е **груба синтактична грешка**.**

## Обичайна грешка при програмиране 6.5

---

Пропускането на **return expression** в метод, връщащ данни (метод, който не е деклариран като **void**) е синтактична грешка.

## Обичайна грешка при програмиране 6.7

---

Формалните аргументи на метод са част от неговите локални променливи и затова *повторната декларацията на променлива в тялото на метода с име на формален параметър е синтактична грешка.*

## 6.6 Машинно представяне на изпълнението на методи

- Stack структура на паметта
  - Last-in, first-out (LIFO- *последния влязъл- излиза пръв*) структура данни
    - Елементи се вмъкват (push) отгоре
    - Елементи се извеждат (pop) отгоре
- Stack структура за изпълнение на методи
  - Наричана още, **execution stack** - Stack структура с изпълнения на методи, **процесорът има директен достъп до тази структура**
  - Когато един метод извиква друг метод, извиквания метод трябва да знае как да върне управлението на викащия метод. Затова **адресът на викащия метод се записва** (push) в execution stack. При последователни извиквания на методи, адресите на викащите методи се записват като *последния влязъл- излиза пръв структура*.
  - Адресите на викащите методи се вмъкват **execution stack**, когато те викат други методи и те се извеждат от **execution stack**, когато виканият метод връща управлението (*виканият метод взема адреса за връщане на управлението от execution stack, след което този адрес се изтрива от execution stack*)

## 6.6 Машинно представяне на изпълнението на методи

- При извикването на метод, неговите локални променливи се вмъкват в част от **execution stack** известна като запис за активиране – “*activation record*”
- При връщане на управлението на викащия метод, записът за активиране се извежда (pop) от **execution stack** и тези локални променливи са недостъпни за изпълняваната програма.
- Когато **една единствена локална** променлива реферира обект, то при извеждане на тази променлива от **execution stack** този обект става недостъпен за програмата и евентуално се изчиства от паметта посредством “garbage collection.” на JVM

# Задачи

1. Едно цяло число се нарича *перфектно*, ако се представя като сума на множителите си. Например, 6 е перфектно число, понеже,  $6 = 1 + 2 + 3$ .
  - Напишете метод  
`public boolean isPerfect( int number)`  
 който определя дали зададения аргумент **number** е перфектно число или не е перфектно число като съответно връща **true** или **false**.
  - Нека са дефинирани *константите* **NUMBER\_PER\_LINE=5** и **NUMBER\_OF\_PRIMES= 10000**
  - Напишете още метод  
`public void showMultipliers( int number)`  
 който извежда табулирано в редове от по **NUMBER\_PER\_LINE** числа множителите на зададения аргумент **number**.
  - Напишете алгоритмите и тяхната UML визуализация за тези два метода
  - Използвайте тези методи в Java приложение, което определя и извежда всички перфектни числа между 1 и **NUMBER\_OF\_PRIMES**. За всяко *перфектно* число открито с **isPerfect()** изведете за проверка и съответните му множители, чрез **showMultipliers()**.

# Задачи

2. Едно цяло число се нарича просто, ако се дели само на 1 и на себе си. Например, 2, 3, 5 и 7 са прости числа, но 4, 6, 8 и 9 не са прости числа.
- Напишете метод  
***public boolean isPrime(int number)***  
който определя дали зададения аргумент е просто число или не е просто число и съответно връща ***true*** или ***false***.
  - Използвайте този метод в **Java приложение** за да намерите и изведете всички прости числа по- малки от ***nLimit = 10,000***, където ***nLimit*** е константа, дефинирана в **Java приложението**. Колко е горната граница за числата по- малки от 10,000, които трябва да тествате?
  - Първоначално може да предположите, че търсената горната граница е ***nLimit / 2***. В действителност тази граница е корен квадратен от ***nLimit***. Защо? Изпълнете програмата с всяка от тези две горни граници и сравнете бързодействието и крайните резултати.

# Задачи

3. Напишете метод *reverseNumber()* , който да взима за аргумент цяло число и да връща цяло число, чиито цифри са в обратен ред на зададения аргумент на метода. Например, ако за реален аргумент на метода е зададено числото **7631**, то методът трябва да връща цялото число **1367**.

Използвайте този метод в Java приложение , което прочита цяло число и извежда резултата от изпълнението на метод *reverseNumber()*

4. Най- големият общ делител на две числа ( *the greatest common divisor* GCD) е най- голямото цяло число, което дели без остатък двете числа.

- Напишете метод *gcd()* който връща най- големият общ делител на две числа.
- Напишете алгоритъма и UML визуализацията за метод *gcd()*
- Използвайте този метод в Java приложение , което прочита две цели числа и извежда резултата от изпълнението на метод *gcd()* заедно с двете въведени числа (за проверка).



# Задачи

5. Най- големият общ делител на две числа  $n1$  и  $n2$  (*the greatest common divisor GCD*) е най- голямото цяло число, което дели без остатък двете числа по следния начин. Първо определете  $d$  като минимума на  $n1$  и  $n2$ , а след това проверете дали в следния ред някое от числата  $d, d-1, d-2, \dots, 2, 1$  е делител едновременно на  $n1$  и  $n2$ . Първото такова число е най- големият общ делител на двете числа  $n1$  и  $n2$ .
- Напишете алгоритъма и UML визуализацията му за така описания способ за пресмятане на най- големият общ делител на две числа  $n1$  и  $n2$
  - Напишете метод `gcd()` който прилага този алгоритъм.
  - Използвайте този метод в Java приложение, което прочита две цели числа и извежда резултата от изпълнението на метод `gcd()` заедно с двете въведени числа (за проверка).

# Задачи

6. В редица случаи се налага използване на меню, за представяне на различни възможности за избор при изпълнение на програма.
- Напишете метод `displayMenu()`, който извежда в текстов формат следното меню
    1. Въведи сума за депозирание в лева
    2. Изведи депозиранията сума в евро
    3. Край
  - Напишете алгоритъма и UML визуализацията за метод, който изобразява това меню, позволява на потребителя да въведе номер на желана опция и я изпълнява, докато не се избере опция 3. След изпълнение на всяка опция да се изчиства текстовия екран преди ново извеждане на менюто. Използвайте константа за курса за преобразуване от лева в евро
  - Напишете метод за `getUserChoice()` за реализация на този алгоритъм
  - Използвайте тези методите `displayMenu()` и `getUserChoice()` в Java приложение.