ВЪВЕДЕНИЕ В ООП

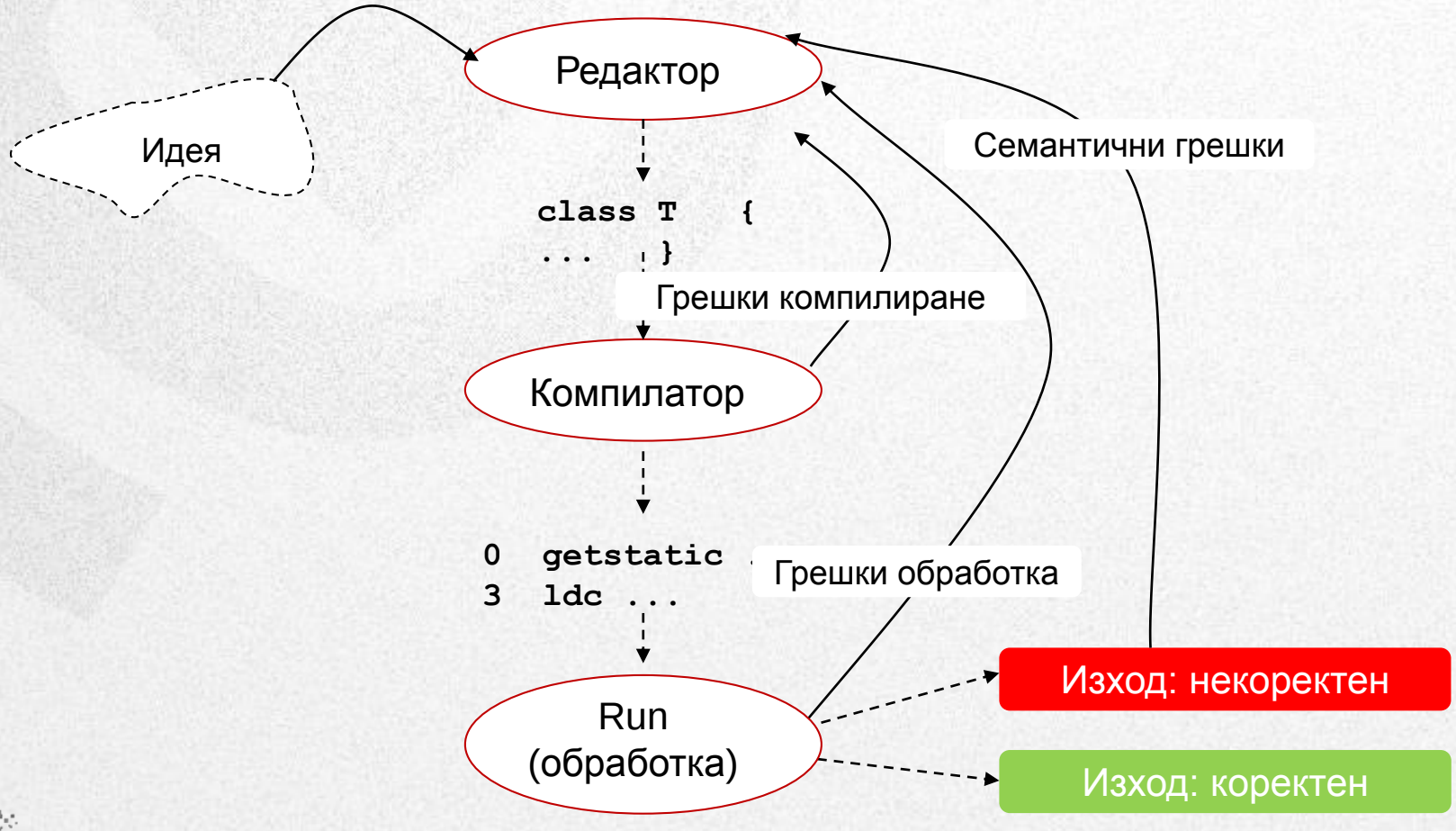
ЛЕКЦИОНЕН КУРС "ПРОГРАМИРАНЕ НА JAVA"

Distributed @learning center

СТРУКТУРА НА ЛЕКЦИЯТА

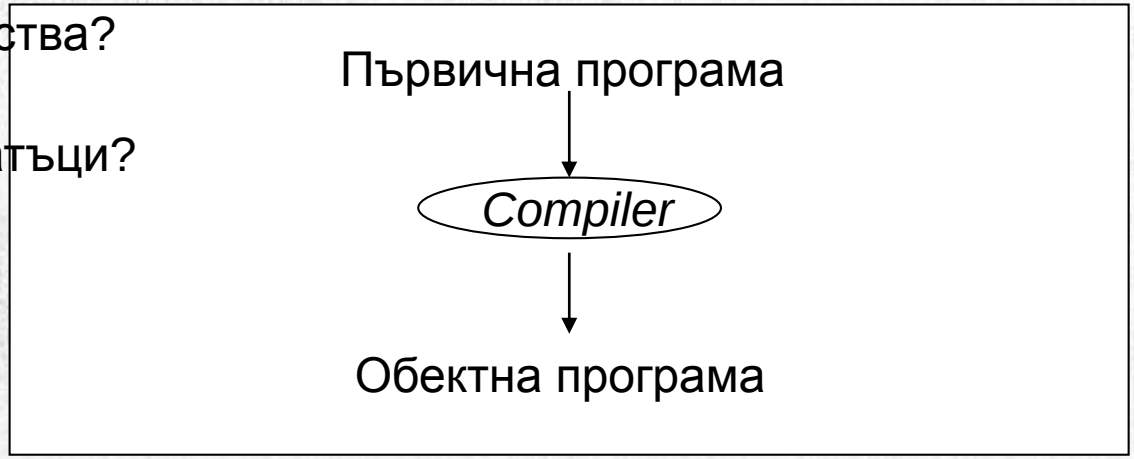
- Разработване на софтуер
- Комплексност
- Овладяване на комплексността
- Капсулиране
- Създаване на обекти
- Работа с обекти
- Примери

РАЗРАБОТВАНЕ НА ПРОГРАМИ



КОМПИЛАЦИЯ

1. Какво е обектна програма?
2. Предимства?
3. Недостатъци?



Машинен (виртуален) код на теоретична машина	Машинен код на реална машина
Интерпретатор (софтуер) ема обработката (виртуална машина)	CPU (хардуер) на машината поема обработката

преносимост

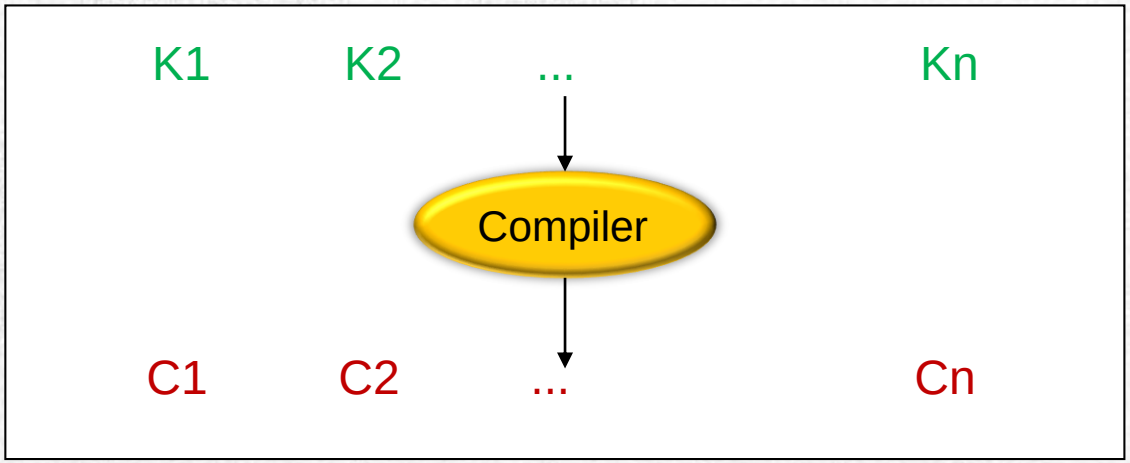
по-бавен

бързина

по-статичен

РАЗДЕЛЕНО КОМПИЛИРАНЕ

Първична програма: множество от компоненти
(компилируеми единици)



Обектна програма: множество от обектни файлове

Компоненти Ki: клас с име 'name', във файл с име 'name.java'

НЯКОИ ОСНОВНИ ПРИНЦИПИ

- След получаване на задача за разработване на софтуер:
  - Не трябва веднага да се захващаме с програмиране
  - Първо добре обмисляне на задачата, обсъждане на неясни постановки ...
- Програмирането е една малка част от цялостния софтуерен развоен процес.
- Също така, особено съществени способности като:
  - Намиране на точна дефиниция на проблемите
  - Систематично търсене на грешки, ...
- Още при програмирането трябва да се мисли за доставката (поддръжката) на софтуера:
  - Четаеми, добре структурирани, добре коментирани програми
- Разработването на софтуер винаги предполага ‘работа в екип’

КОМПЛЕКСНОСТ НА СОФТУЕРА

- Софтуерните системи принадлежат към най-комплексните създания на човека:
  - Структурите и поведението на големите системи в общия случай не са обозрими;
  - Те не могат да бъдат напълно разбрани както в началото при развоя, така също и в края при тестването,експлоатацията и поддръжката.

- Решаващата характеристика на индустриално използвания софтуер е, че за отделния разработчик е много трудно (дори невъзможно) да рабере всички тънкости на развоя
  - Просто казано, комплексността на такива системи надхвърля възможностите на човешкия интелект

СВОЙСТВА НА СОФТУЕРА

- **Няма изхабяване**
  - при многократна употреба
- **Лесен за копиране**
  - също грешките
- **Остарява**
  - софтуерът постоянно се приспособява
- **Дълго в употреба**
- **Трудно измерим**
  - метрики: качество, количество
- **Изключително комплексен**
  - в практиката

Софтуер =
Програми, данни, документация

КАЧЕСТВЕНИ КРИТЕРИИ ЗА СОФТУЕРНИТЕ ПРОДУКТИ

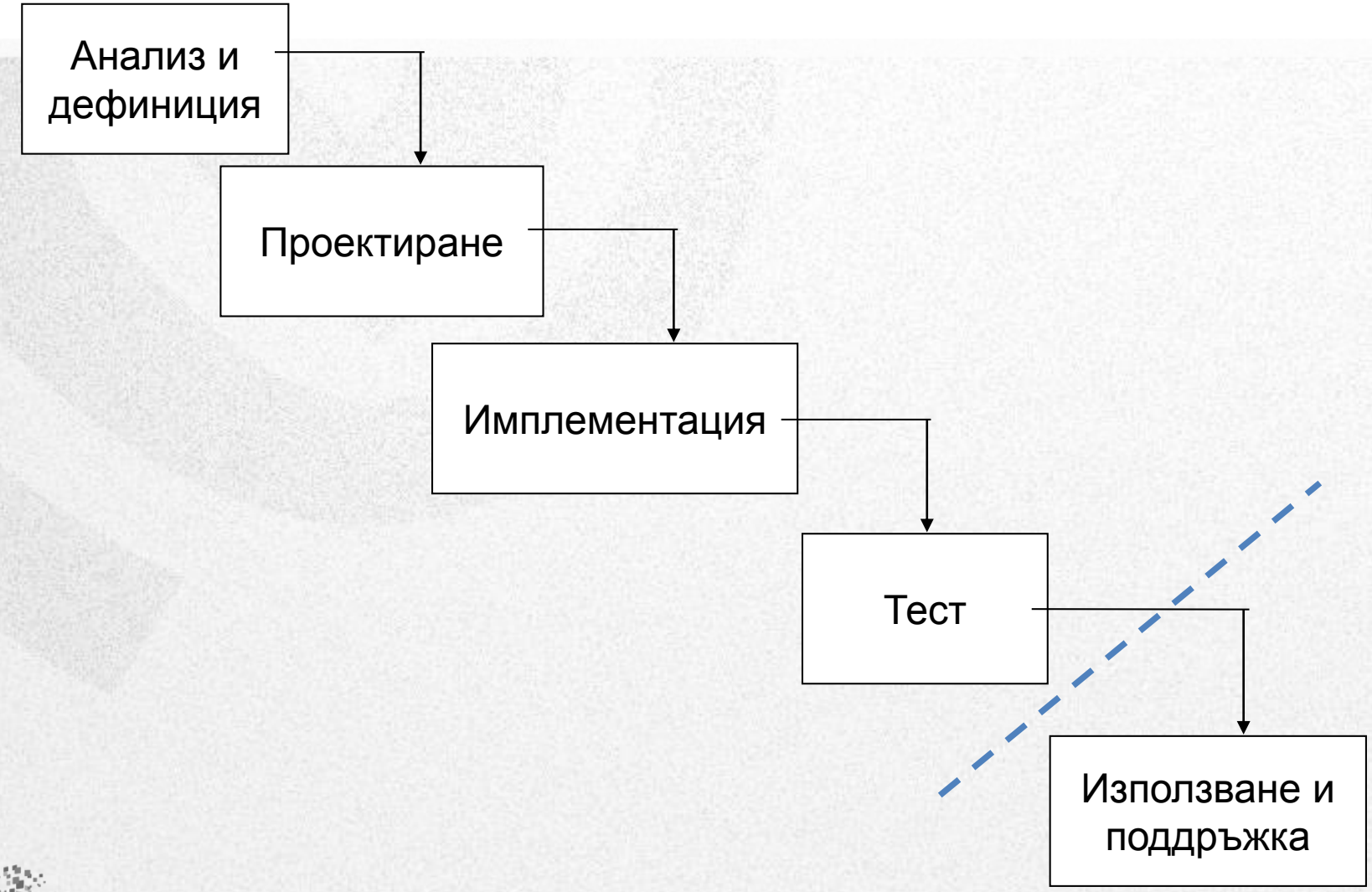
- Коректност
- Стабилност: напр. при обработка на грешки
- Ефективност
- Удобен за потребителя
- Документираност:описание на програмите
- Модифицируемост
- Четаемост (разбираемост): коментари, смислен избор на означения, форматиране ...
- Многократна използваемост
- Модулност: декомпозиран на модули / компоненти
- Преносимост: върху различни компютърни конфигурации
- Интегрируемост: защита срещу неправомерен достъп
- ...

МОДЕЛИ НА РАЗРАБОТВАНЕ НА СОФТУЕР

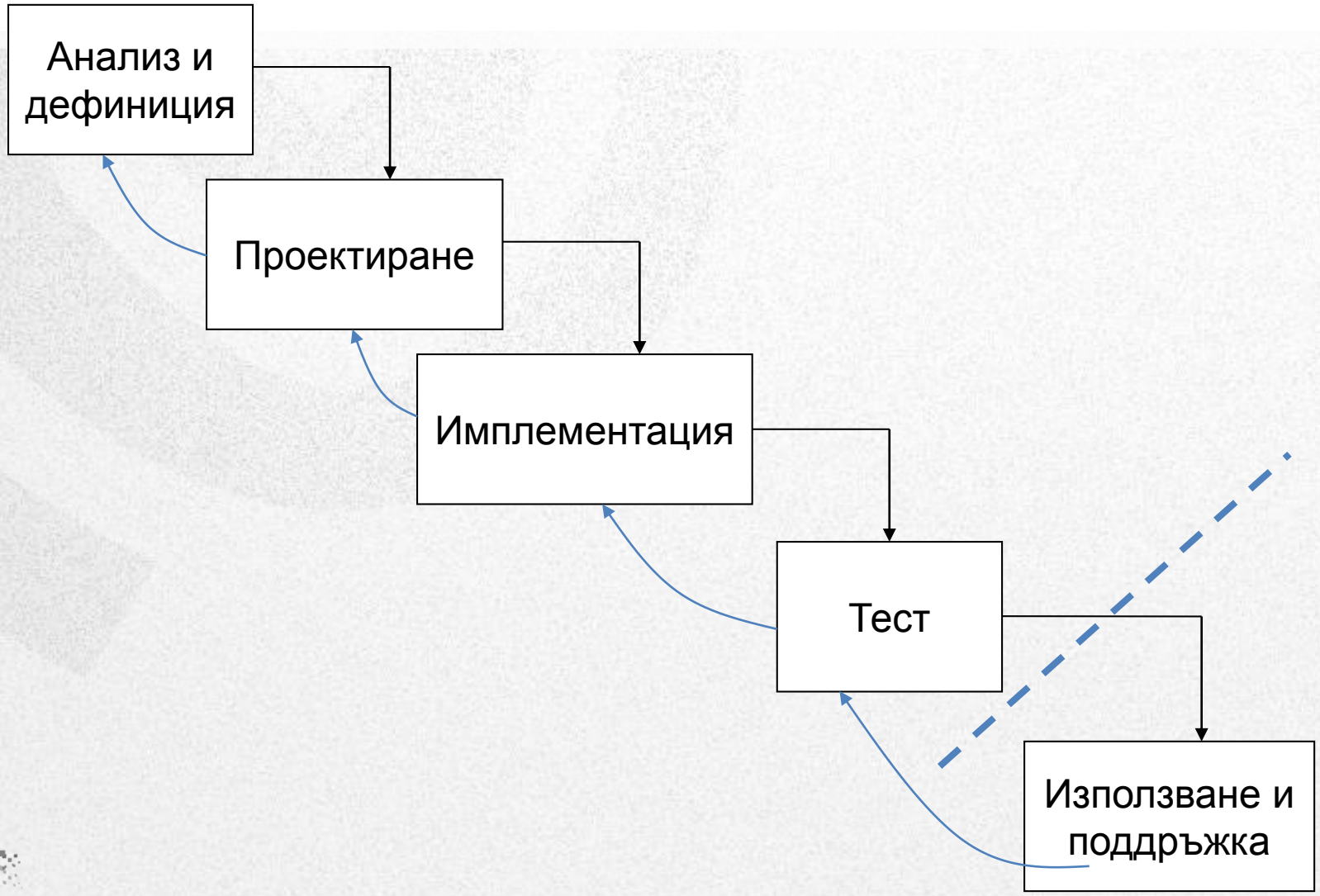
- Алтернативни понятия:
  - Модели на подход
  - Фазови модели
  - Модели на жизнен цикъл
- Проблеми:
  - Грешен софтуер
  - Забавен развой
  - Загуба на средства
  - ...

Цел: систематичен развой на софтуер – съществено при разработване на комплексни системи
Опит: без систематичен подход няма качествен високостойностен софтуер

КЛАСИЧЕСКИ ВОДОПАДЕН МОДЕЛ



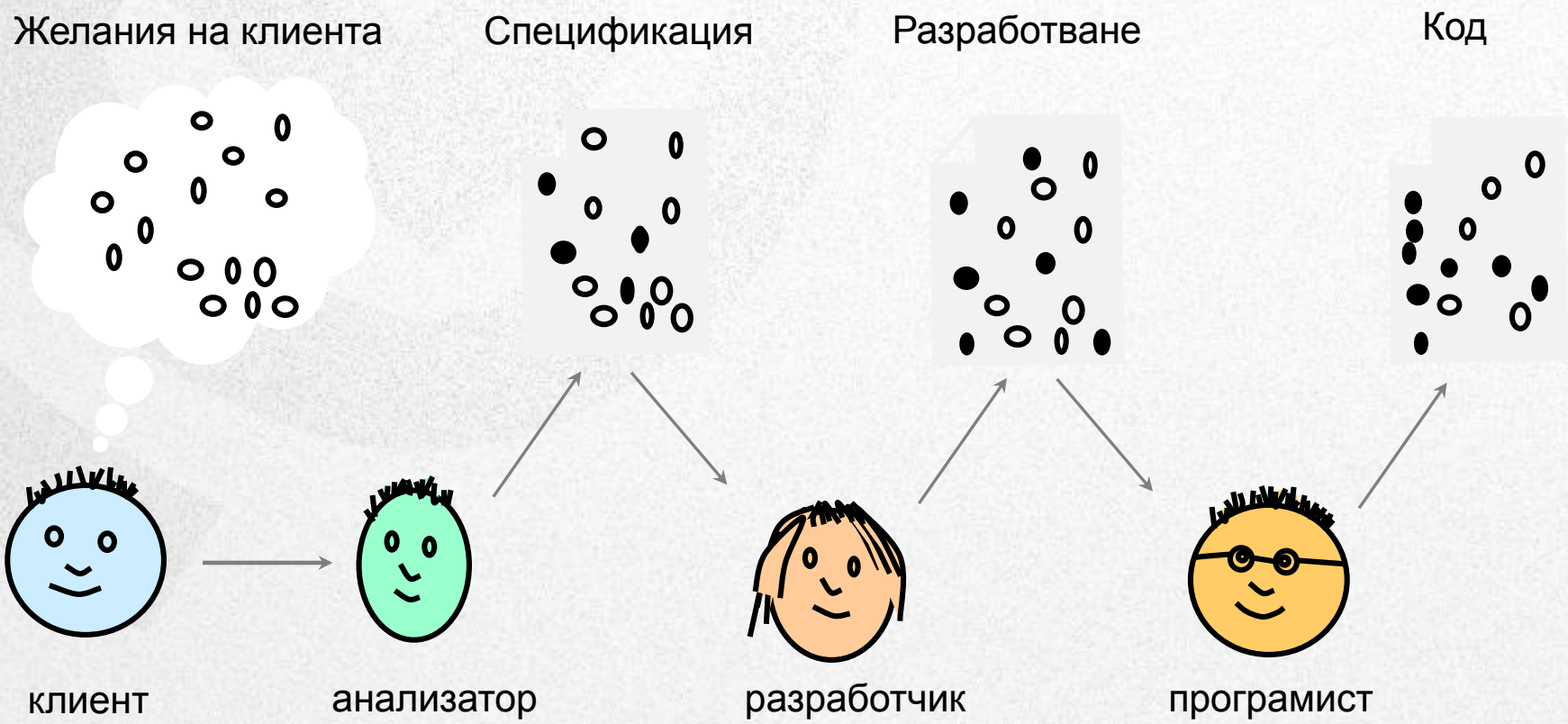
ИТЕРАТИВЕН ФАЗОВ МОДЕЛ



РАЗРАБОТВАНЕ НА СОФТУЕР: ФАЗИ И ПРОДУКТИ

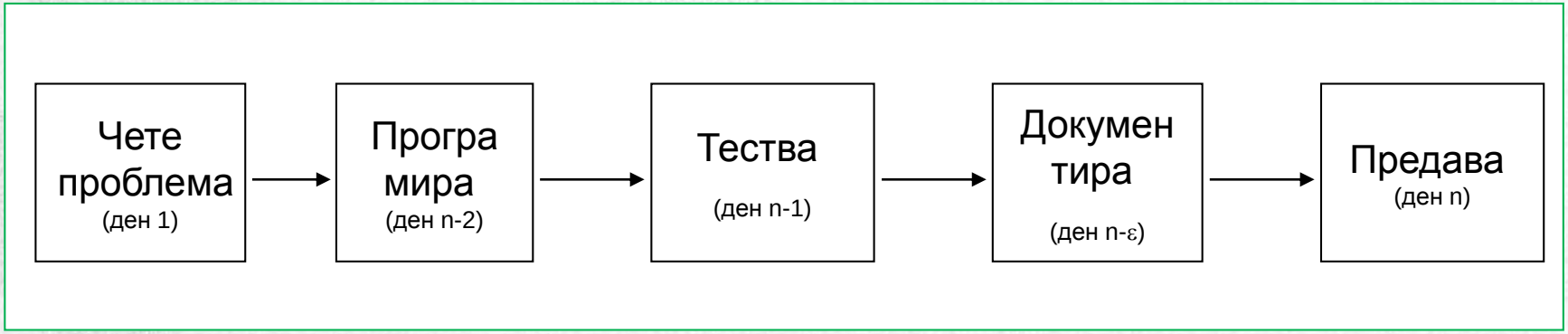
- **Анализ и дефиниция**
  Анализ на проблема + дефиниране на изискванията към софтуерния продукт
  Предмет: външното поведение на софтуерната система
  Поръчител ↔ изпълнител
  ➔ Дефиниция на продукта (системна спецификация, дефиниця на изискванията, техническо задание)
- **Разработване (проектиране)**
  Установяване на структура, изграждане, компоненти на софтуера и техните връзки
  ➔ софтуерна архитектура
- **Имплементация**
  "Изпълнение" на софтуерната архитектура: програмиране на компонентите
  ➔ програма
- **Тест**
  тест на компонентите, тест на тяхната интеграция (системен тест)
  ➔ тестови протоколи

ДОКУМЕНТИ НА РАЗВОЙНИЯ ПРОЦЕС



СТУДЕНТСКО РАЗРАБОТВАНЕ НА СОФТУЕР

Обикновено това е представата на студентите за жизнения цикъл на софтуер!



МОДЕЛИ: ПРЕГЛЕД

- Класически фазов модел (водопаден модел)
- Итеративен фазов модел (жизнен цикъл)
- Спирален модел
- Прототипиране (еволюционно разработване)
- Трансформационно разработване
- Използване на многократно използваеми компоненти

Лекционен курс “Софтуерно инженерство"

ОСНОВНИ ХАРАКТЕРИСТИКИ НА ООП

- Основен проблем на разработване на софтуер: **комплексност**
- ООП възниква за решаване на този проблем
- Основни характеристики на ООП:
  - Капсулиране
  - Многократна използваемост
    - Поведението на обектите е генетично: т.е. те могат да бъдат използвани в различни ситуации
  - Наследяване
    - Обектите са основа, от която водят началото си други обекти

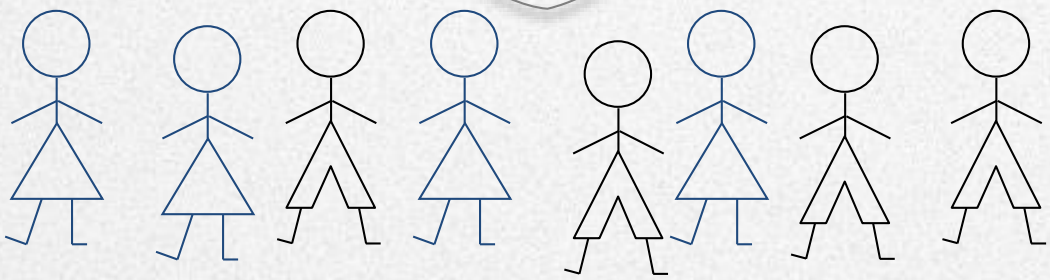
ЕДНА РЕАЛНА ИСТОРИЯ ...

Софтуерът може да бъде много комплексен ...

Software

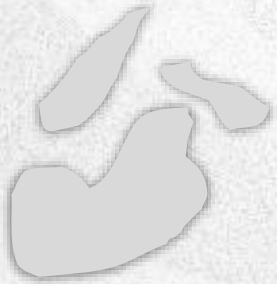
Разработчици: Никога не можем да го създадем ...

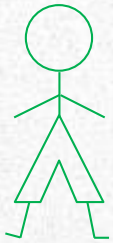
Разработчици: Не го разбираме

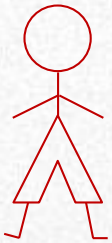


ОВЛАДЯВАНЕ НА КОМПЛЕКСНОСТТА: РАЗДЕЛЯЙ И ВЛАДЕЙ

Разлагане на модули

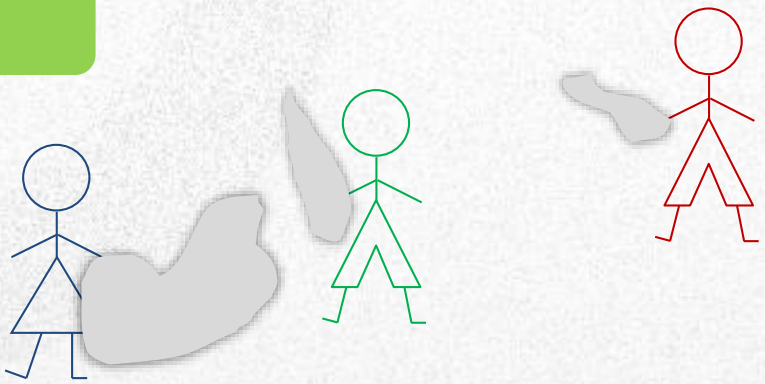






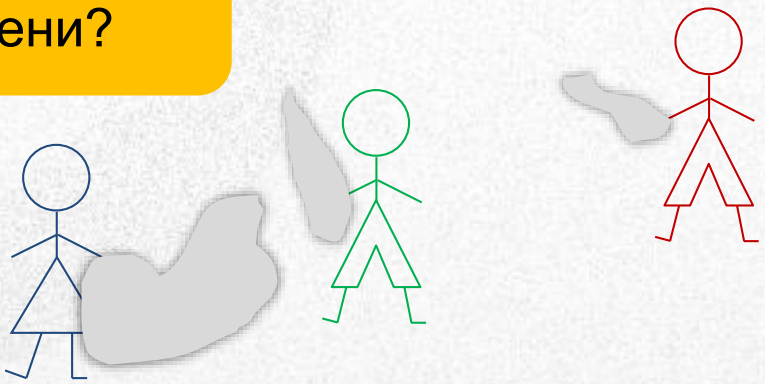
ЕДНА ГОДИНА ПО-КЪСНО …

Готови сме!



ЕДНА ГОДИНА ПО-КЪСНО …

Нещата не изглеждат така, както са замислени?



ДЕКОМПОЗИРАНЕ

- Софтуерните системи не могат да бъдат напълно разбрани:
  - Както предварително при развоя
  - Така също и впоследствие, при използването им
- Първи принцип за овладяване комплексността на разработването на софтуер:
  - **декомпозиция**

ОВЛАДЯВАНЕ НА КОМПЛЕКСНОСТТА: ДЕКОМПОЗИЦИЯ

- Разлагане на софтуера на модули
- Всеки модул: поединично обхващаем
- Модули:
  - Разработване независимо от останалата част на системата
  - Семантично:
    - Модулите съответстват на подзадачи

ОВЛАДЯВАНЕ НА КОМПЛЕКСНОСТТА: АБСТРАКЦИЯ

- Софтуерните системи не могат да бъдат напълно разбрани:
  - Както предварително при развоя
  - Така също и впоследствие, при използването им
- Втори принцип за овладяване комплексността на разработването на софтуер:
  - **абстракция**

ОВЛАДЯВАНЕ НА КОМПЛЕКСНОСТТА: АБСТРАКЦИЯ

- Модулите са **абстракции**
- Останалата част от софтуерната система познава само:
  - Външното поведение на модулите
  - Не обаче детайли на реализацията

Интерфейс на модула

ЕДНА БАНАЛНА КОЛЕДНА ИСТОРИЯ

Какво се случва след като децата легнат да спят на Коледа? ....

Дядо Коледа е един весел старец, облечен в червен кожух, пристига на шейна, теглена от елени, ...

На Коледа през нощта Дядото остава подаръци под елхата.

От своя страна преди да легнат да спят децата му оставят шоколад и нещо топло за пиене (напр. мляко) ...

След Коледа (до следващата) Дядото се възстановява и подготвя списък с подаръци за следващата година ...

След като децата са заспали родителите им започват да изпълняват ролята на Дядо Коледа ....

Майката подрежда подаръците под елхата....

Бащата излива млякото обратно в бутилката (чашата трябва да бъде намерена празна, понеже ... Дядо Коледа е изпил млякото ...)...

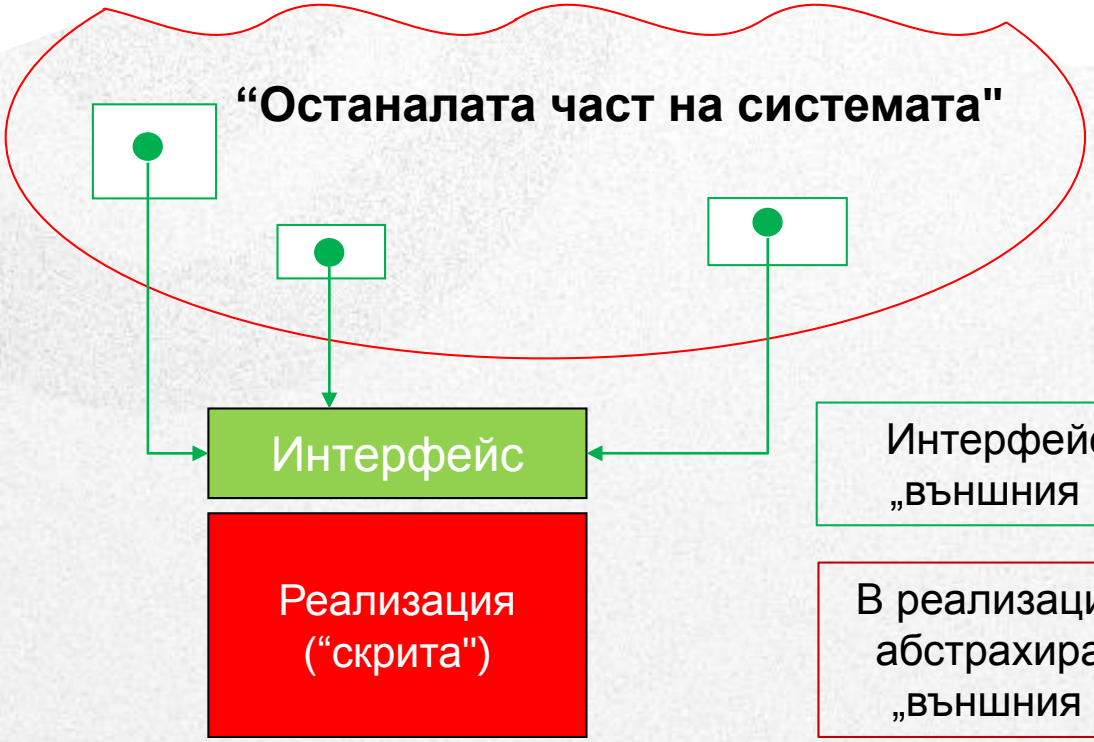
Отваря шоколада и ... си налива чаша уиски (шоколадът трябва да го няма, понеже ... Дядо Коледа го е изял ...) ...

КАКЪВ Е ИЗВОДЪТ ОТ КОЛЕДНАТА ИСТОРИЯ?

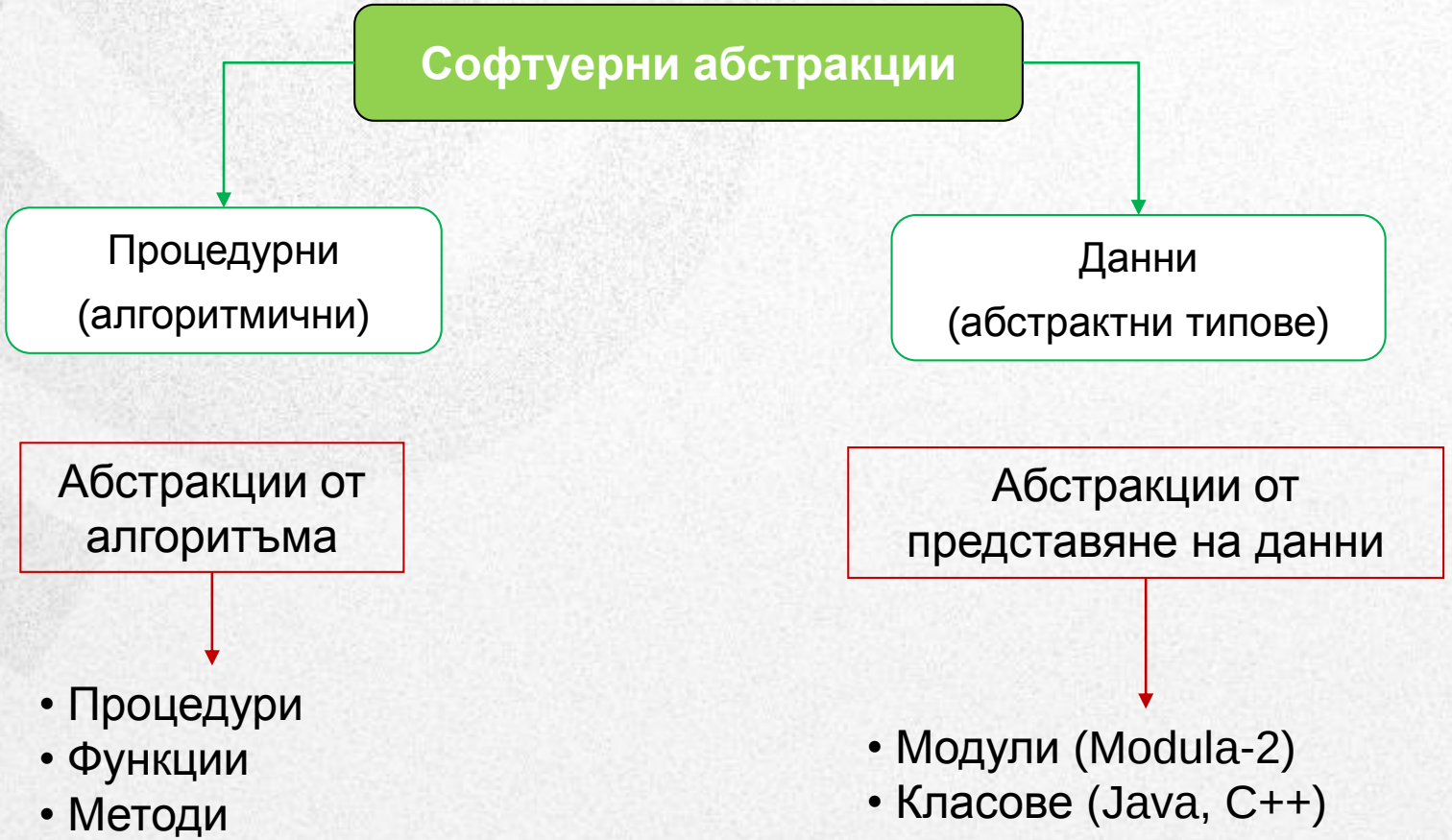
1. Без декомпозиция и абстракция …

Коледата невъзможна !

МОДУЛИ


“Останалата част на системата"
Интерфейс
Реализация
(“скрита")
Интерфейс към „външния свят“
В реализацията се абстрахираме от „външния свят“

ОСНОВНИ СОФТУЕРНИТЕ АБСТРАКЦИИ



ИМПЕРАТИВНО ПРОГРАМИРАНЕ

Ориентирано към описание на алгоритми

Java:

```
class All {
  public static void main (...) {
     x = Keyboard.readDouble();

     Алгоритъм:
           отделни оператори ...

     System.out.println(y);
     System.out.println(z);
  }
}
```

Вход

Разлагане

Изход

ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ: КЛАСОВЕ

- Основен проблем в програмирането
  - Дефиниране на нови типове на обекти
- С помощта на конструкцията **class** можем да дефинираме нови типове от обекти, като:
  - Атрибути
  - Операции
- Два аспекта можем да разглеждаме разделено:
  - **Използване на класове**
  - **Дефиниране на класове**

СИНТАКСИС

```
class name   {

        declarations

        constructor definitions

        method definitions

}
```

Символни константи и променливи

Код за създаване и инициализиране на обекти

Код за манипулиране на тези обекти

МЕТОДИ

- Капсулирането на код се извършва под формата на **методи**
- Създаването на методи е свързано с императивното програмиране
  - Алгоритми/подалгоритми
- Основни проблеми:
  - Декларация, извикване на методи
  - Актуални, формални параметри
  - Параметри по стойност, референция

ПРОЦЕДУРНА АБСТРАКЦИЯ: МЕТОДИ

“Останалата част на системата ":

```
fac = faculty (10);
```

```
public static int faculty (int n) {
```

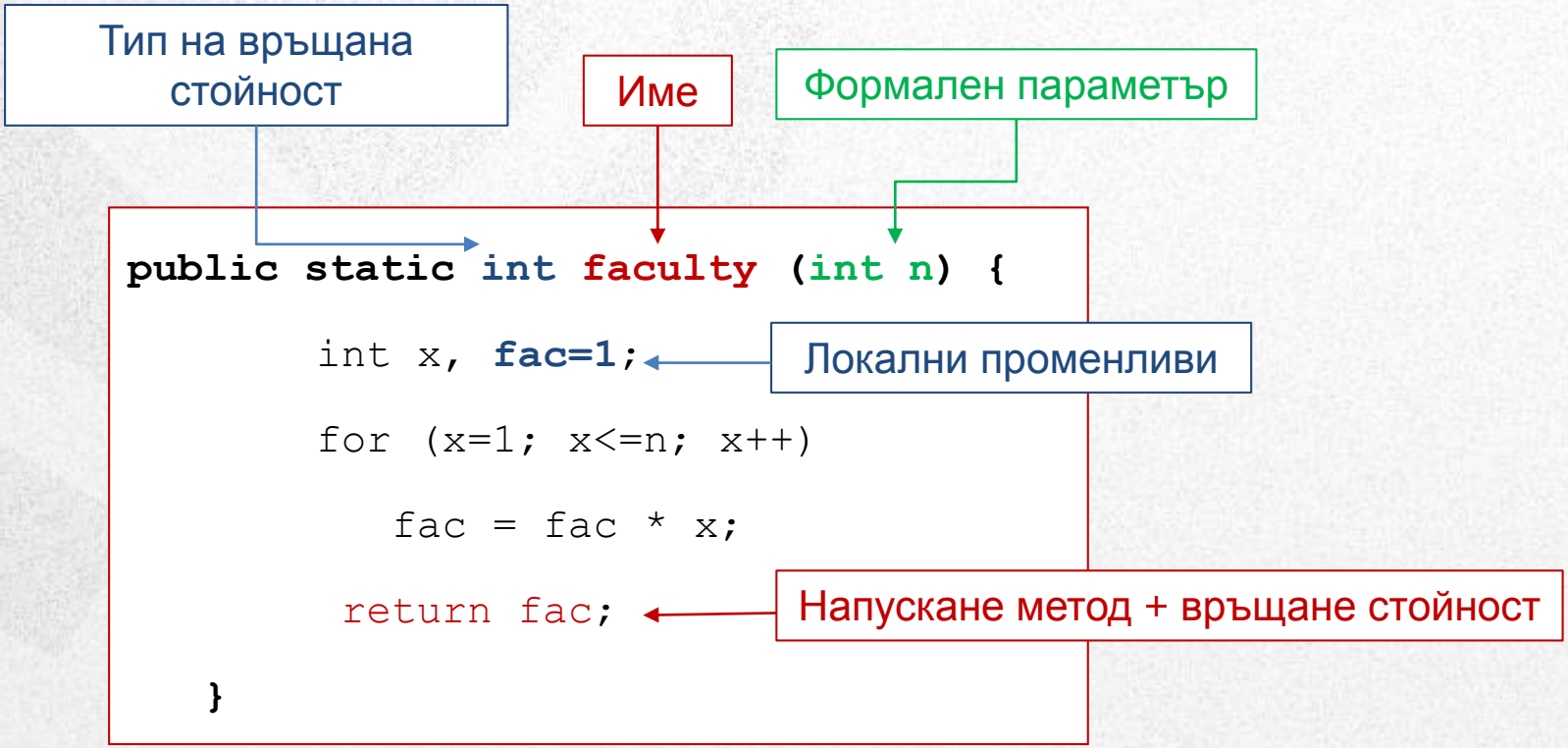
```
        int x , fac = 1 ;

        for (x = 1; x <= n ; x++)
                fac = fac * x;
        return fac ;
  }
```

ТЕСТ: FACULTY

1 Коя е „останалата част на системата“?

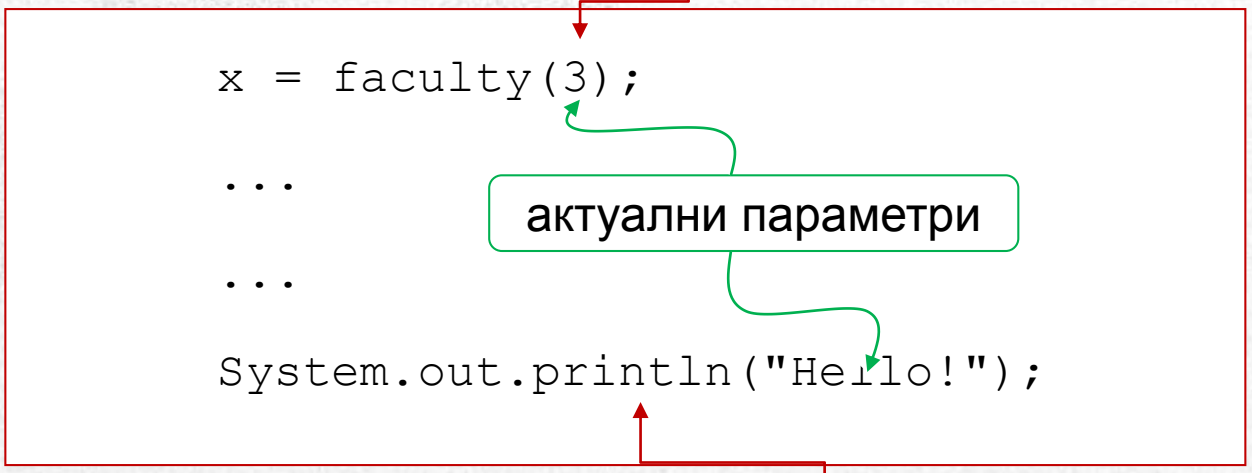
```
class Faculty {

   public static int faculty (int n) {
        int x, fac = 1;

        for (x=1; x<=n; x++)
           fac = fac * x;
        return fac;
   }

   public static void main (String[] args) {
        int x, y;

        x = faculty(3);
        System.out.println("3! = " + x);
        y = faculty(5);
        System.out.println("5! = " + y);
   }
}
```

ДЕКЛАРАЦИЯ НА МЕТОД: ОПИСАНИЕ НА АЛГОРИТЪМА



ИЗВИКВАНЕ НА МЕТОД: АКТИВИРАНЕ НА АЛГОРИТЪМА

1. Случай:
С целеви тип (напр. int) като израз



2. Случай:
Без целеви тип (=void) като оператор

ПРЕДАВАНЕ НА ПАРАМЕТРИ В JAVA

```
faculty(x);
System.out.println("...");
```

- **Основно:** параметри-стойности
  всички елементарни типове
- **Обекти:** параметри-референции
  също така масиви(arrays)

Вид на предаването на параметри в Java:

- Зависи от типа на параметъра
- Не съществува ключова дума за различаване (напр. VAR)

ОБОБЩЕНИЕ: МЕТОДИ

В Java: **капсулирането** на кода се нарича **методи**

- Видове методи
  - На инстанции
  - На класове
    - main – винаги метод на клас
- Интерфейс
  - Методи
  - Конструктори

ПРИМЕР ЗА JAVA-ПРОГРАМА:TIME

- Ще разгледаме един по-сложен пример
- Последователно ще разширяваме примера за демонстриране и обясняване различни съществени механизми на обектите
- Класове:
  - class Time
  - class TimePlan

КЛАС:TIME

```
class Time {

   декларации

   public Time (int h,int m) { … }

   public Time addMinutes (int m) { … }

   public void printTime () { … }
}
```

- Широко използваем тип данни: **Time**
- Стойностите му представят времето в един отделен ден:
  - Час: integer, 0 .. 23
  - Минута: integer, 0 .. 59

- **Интерфейс:** публичните методи и конструкторите

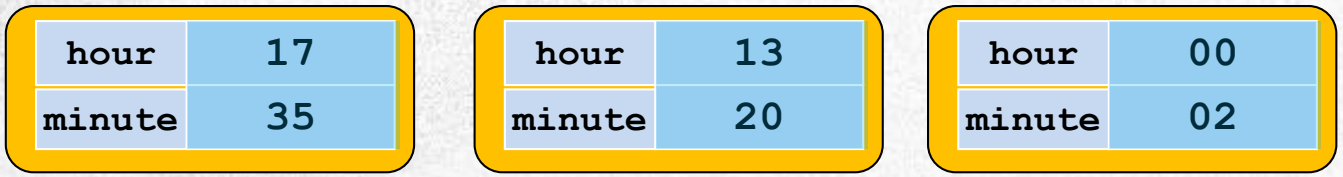
- **Характеристика на ООП:** разделено разглеждане на два отделни аспекта:
  - Използване на класа (клиент)
  - Разработване на класа

ПРАВА НА КЛИЕНТИТЕ

- Да декларират променливи от тип class
- Да създават обекти на класа, използвайки конструктури
- Да изпращат съобщения към обектите, използвайки методите на инстанции, дефинирани в класа
- Да познават публичния интерфейс на класа
  - Имената на методите на инстанции, броя и типа на параметрите, типа на резултатите
- Да знаят кои методи на инстанции променят обектите

ПРИМЕРИ ЗА TIME-ОБЕКТИ

Heap memory



КЛИЕНТИ НА КЛАСОВЕ: ИЗПОЛЗВАНЕ И ДЕФИНИРАНЕ

- Клиенти на един клас
  - Методите, които използват този клас
- Класовете и клиентите имат различни права
- Създаването на един обект става поетапно
  - Установяване на контейнер
    - Декларация на променлива
  - Присвояване указател към контейнера като стойност на променливата
  - Разполагане на обект
    - Само с извикване на конструктор
      - Посредством new
- Възможно е декларирането и конструирането да бъдат извършени в един оператор

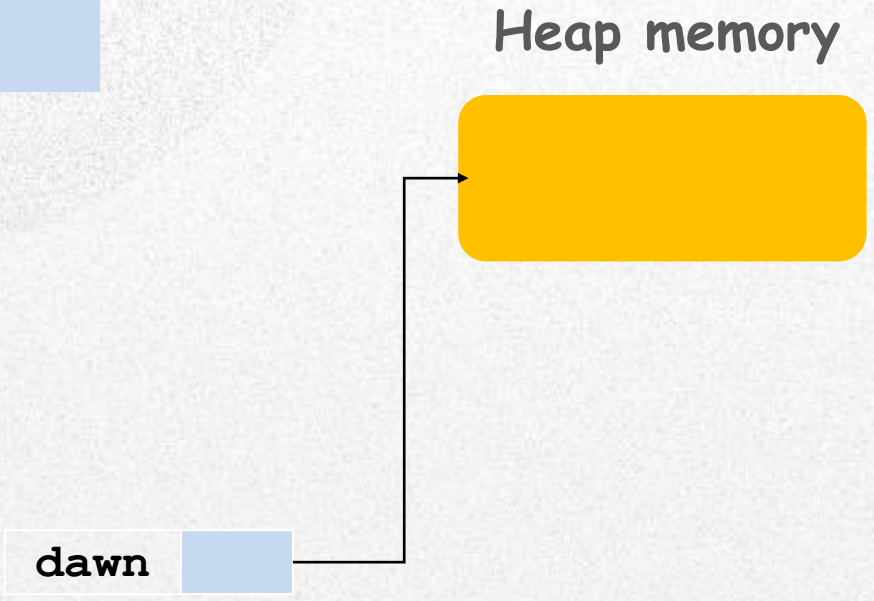
СЪЗДАВАНЕ НА ОБЕКТИ

```
Time dawn;
dawn = new Time(5,35);
```

Heap memory

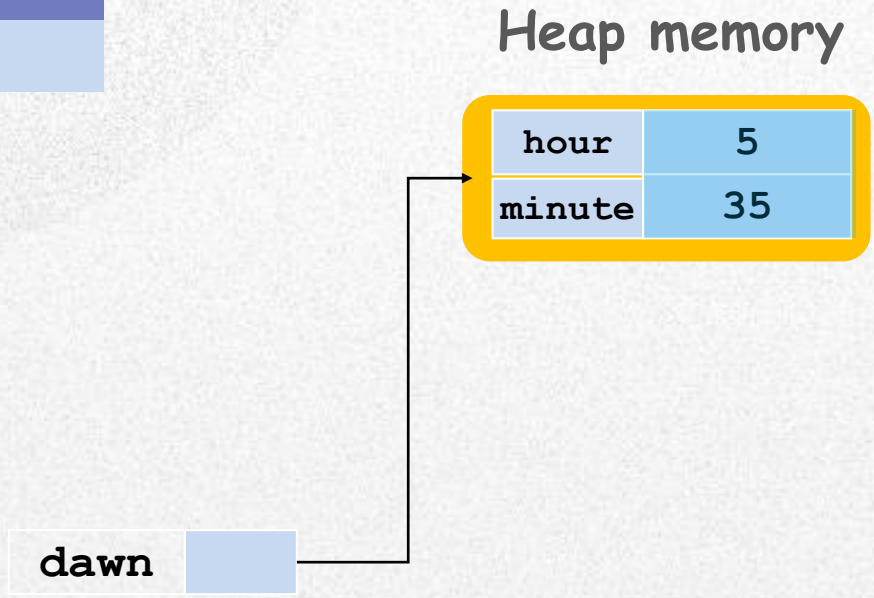
СЪЗДАВАНЕ НА ОБЕКТИ

```
Time dawn;
dawn = new Time(5,35);
```



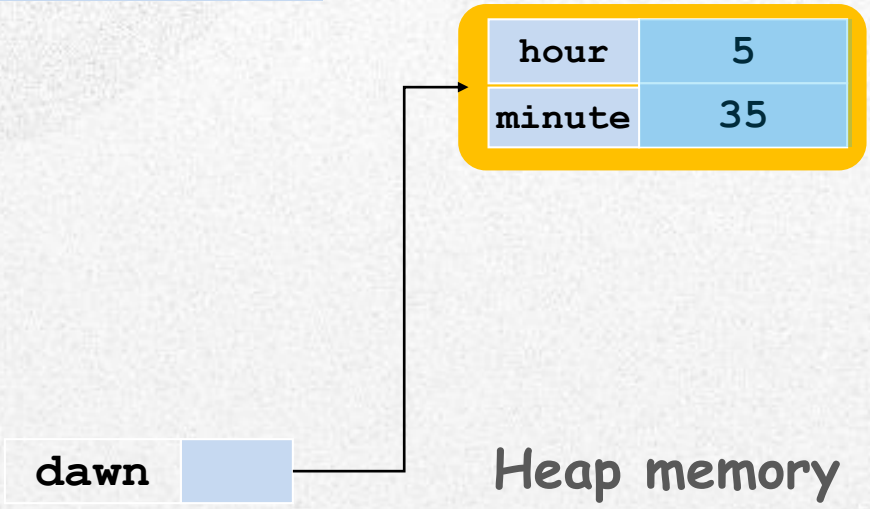
СЪЗДАВАНЕ НА ОБЕКТИ

```
Time dawn;
dawn = new Time(5,35);
```



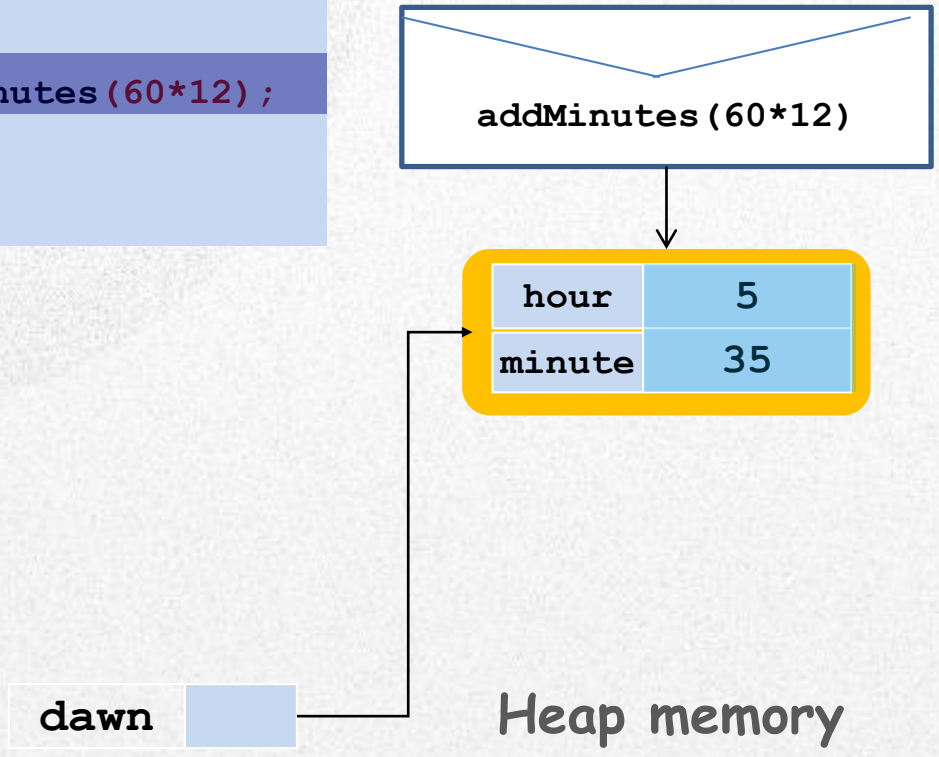
РАБОТА С ОБЕКТИ

```
Time dusk = dawn.addMinutes(60*12);
dusk.printTime();
```



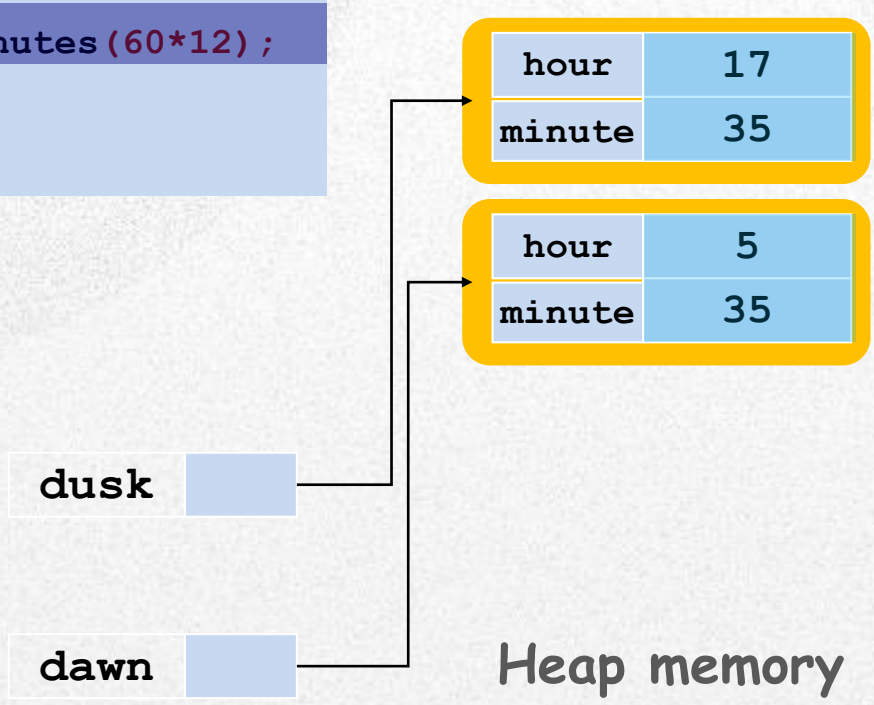
РАБОТА С ОБЕКТИ

```
Time dusk = dawn.addMinutes(60*12);
dusk.printTime();
```



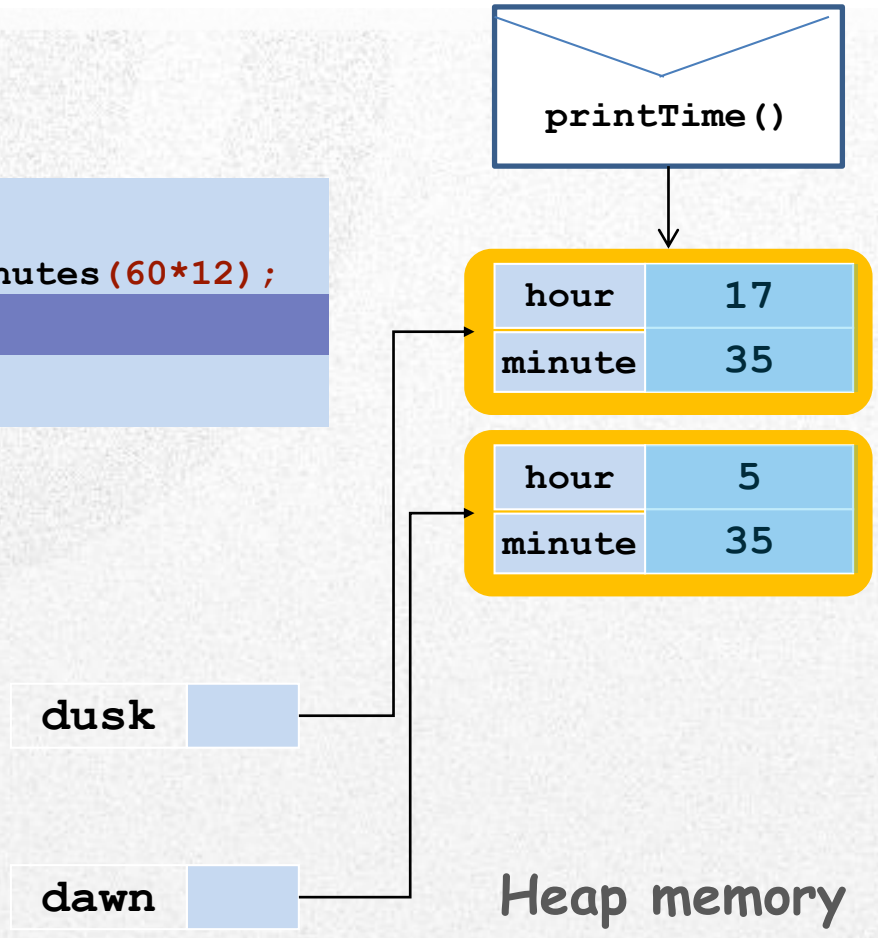
РАБОТА С ОБЕКТИ

```
Time dusk = dawn.addMinutes(60*12);
dusk.printTime();
```



РАБОТА С ОБЕКТИ

```
Time dusk = dawn.addMinutes(60*12);
dusk.printTime();
```



КЛАС:TIME

```
class Time {
   private int hour, minute;
   public Time (int h,int m) { hour = h; minute = m; }
   public Time addMinutes (int m) {
    int totalMinutes = (60*hour + minute + m)%(24*60);
    if (totalMinutes < 0)
          totalMinutes = totalMinutes + 24*60;
    return new Time(totalMinutes/60, totalMinutes%60);
   }
   public void printTime () {
    if ((hour == 0) && (minute == 0))
      System.out.print("midnight");
    else if ((hour == 12) && (minute == 0))
      System.out.print("noon  ");
    else {
      if      (hour == 0) System.out.print(12);
      else if (hour > 12) System.out.print(hour-12);
      else                System.out.print(hour);

      if (minute < 10) System.out.print(":0" + minute);
      else             System.out.print(":" + minute);

      if (hour < 12) System.out.print("AM");
      else           System.out.print("PM");
    }
  }
}
```

ПРАВА НА КЛАСОВЕТЕ

- Да дефинират публичния интерфейс на класа
- Да скриват от клиентите всички детайли на имплементацията
- Да защитават “вътрешните” данни от достъп на клиентите
- Да променят детайли на имплементацията по всяко време запазвайки публичния интерфейс
  - Ако се налага промена на интерфейса - съгласувано с клиентите

TIME: ПРОМЕНЛИВИ НА ИНСТАНЦИИ

1 Характеристика на променливите?

```
class Time {

    private int hour, minute;

    public Time (int h,int m) { …  }

    public Time addMinutes (int m) {

        …
    }

    public void printTime () {

        …
    }
}
```

- **Модификатори:** различни възможности:
  - **public** – могат да бъдат достъпни и променяни от код, извън класа
  - **private** - могат да бъдат достъпни и променяни само от методи на класа (обикновено в ООП)
  - други – по-късно в лекционния курс

TIME: КОНСТРУКТОРИ

1 Какво е конструктор и защо конструктори?

```
class Time {

   private int hour, minute;

   public Time (int h,int m) { …  }

   public Time addMinutes (int m) {

        …
   }

   public void printTime () {

        …
   }
}
```

- **Конструктор:** специален метод
  - Също име като класа
  - Няма тип на връщаната стойност (дори void)
  - Могат да бъдат декларирани като public (обикновено) или private (Java за напреднали)

- **Използване:** инициализация на нови обекти от съответния клас

TIME: ПОВЕЧЕ КОНСТРУКТОРИ

```
class Time {

   декларации

   public Time (int h,int m) { hour = h; minute = m;}

   public Time ( ) {hour = 0; minute = 0;}

   public Time (int m) {hour = m/60; minute = m%60;}

   public Time addMinutes (int m) { … }

   public void printTime () { … }
}
```

- Смислено е да се използват повече от един конструктори
- Различават се по броя или типа на параметрите

Пример:

```
Time t1 = new Time();
     t2 = new Time(720);
     t3 = new Time(525);
```

Еквивалентен на:

```
Time t1 = new Time(0, 0);
     t2 = new Time(12, 0);
     t3 = new Time(8, 45);
```

TIME: МЕТОДИ НА ИНСТАНЦИИ

1. Колко и какви са параметрите на двата метода?

```
class Time {

   private int hour, minute;

   public Time (int h,int m) { …  }

   public Time addMinutes (int m) {

        …
   }

   public void printTime () {

        …
   }
}
```

Time *receiver*

Публични методи, т.е. принадлежат към интерфейса на класа

За всеки метод съществува неявен (вътрешен) параметър от типа на съответния клас. Съдържа получателя на съобщението, когато се извиква метода

TIME: МЕТОД ADDMINUTES

1 Какво прави метода?

```
class Time {

   private int hour, minute;

   public Time (int h,int m) { …  }

   public Time addMinutes (int m) {
     int totalMinutes = (60*hour + minute + m)%(24*60);
     if (totalMinutes < 0)
        totalMinutes = totalMinutes + 24*60;
    return new Time(totalMinutes/60, totalMinutes%60);
   }

   public void printTime () {

       …
   }
}
```

m може да бъде:
- много голяма стойност
- отрицателна стойност

TIME: МЕТОД PRINTTIME

1 Какво прави метода?

```
class Time {

   private int hour, minute;
   public Time (int h,int m) { …  }
   public Time addMinutes (int m) { … }

   public void printTime () {
     if ((hour == 0) && (minute == 0))        System.out.print("midnight");
     else if ((hour == 12) && (minute == 0)) System.out.print("noon  ");
          else {
            if      (hour == 0) System.out.print(12);
            else if (hour > 12) System.out.print(hour-12);
                 else           System.out.print(hour);
     if (minute < 10) System.out.print(":0" + minute);
     else             System.out.print(":" + minute);
     if (hour < 12)   System.out.print("AM");
     else             System.out.print("PM");
    }
  }
}
```

UK стандарт за представяне на времето

РАЗШИРЕНИЕ НА КЛАСА TIME

- Искаме да добавим повече функционалност в класа
- Като нови методи
  - subMinutes
  - priorTo
  - after

МЕТОД SUBMINUTES

- Една възможност
  - *t.subMinutes(i)* еквивалентен на *t.addMinutes(-i)*
- Ако искаме повече удобство за потребителите

1 Проблем?

```
public Time subtractMinutes(int m)   {
        return receiver-of-subtractMinutes.addMinutes(-m);
}
```

time обект, който получава извикването

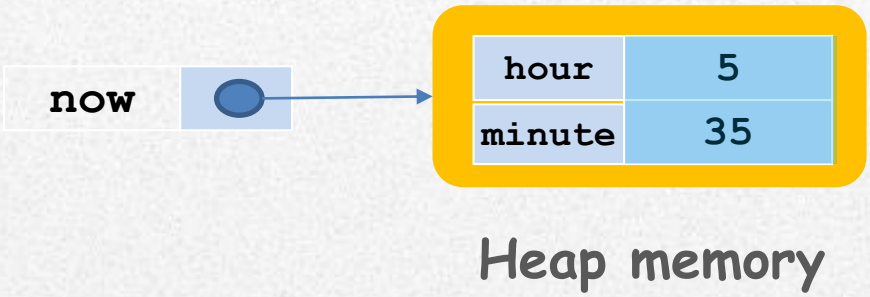
МЕТОД SUBMINUTES

- Нямаме възможност да именуваме *receiver* на метода
- Как можем да завършим кода?
  - В тялото на един метод можем да извикаме друг метод от същия клас без явен получател
  - Методът получава същия получател като този, в който се съдържа

```
public Time subtractMinutes(int m)   {
        return addMinutes(-m);
}
```

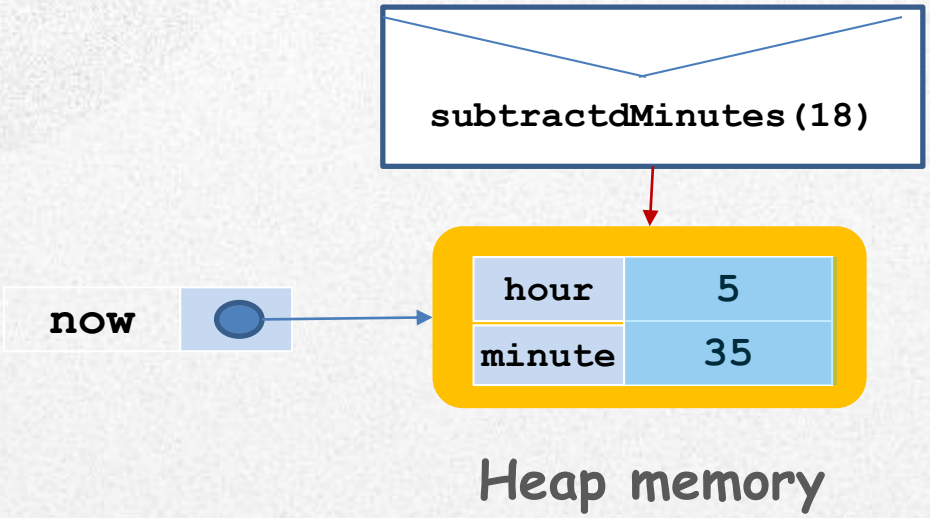
ИЗПОЛЗВАНЕ НА МЕТОДА

```
now = now.subtractMinutes(18);
```



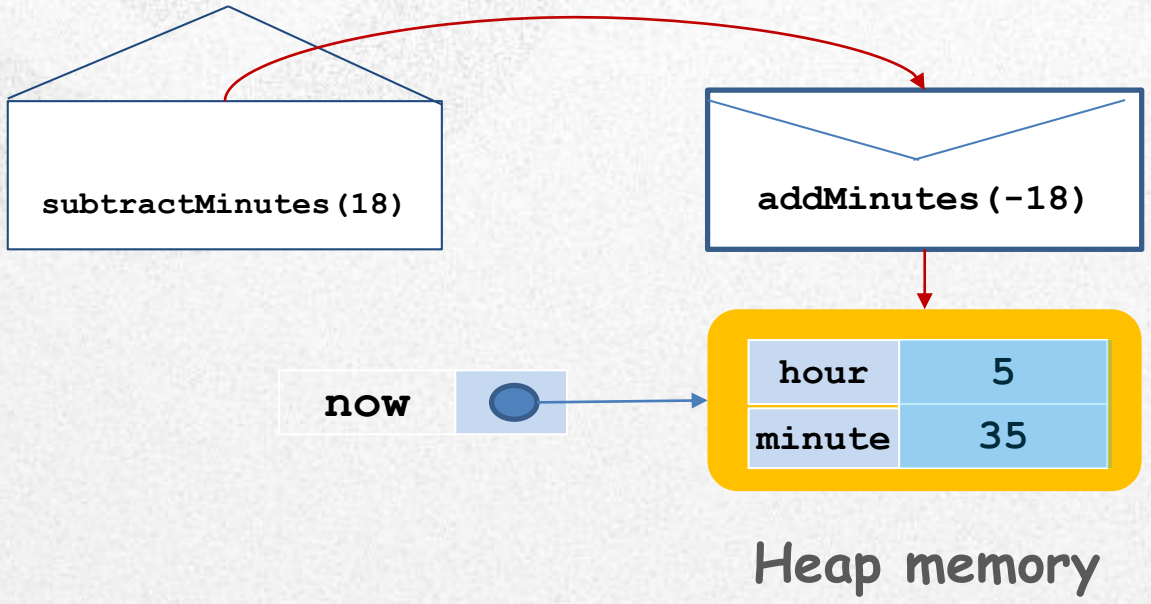
ИЗПОЛЗВАНЕ НА МЕТОДА

```
now = now.subtractMinutes(18);
```



ИЗПОЛЗВАНЕ НА МЕТОДА

```
now = now.subtractMinutes(18);
```



PRIORTO

- Методът сравнява две времена, като проверява дали актуалното (в обекта, където е зададен метода) е "преди" друго (зададено в аргумента)
- При кодиране на метода трябва да можем да виждаме инстанция на променливата t.
- Съществено: знаем как да видим променливите на инстанции само на получателя, но t не е получател, а аргумент
  - Получателят не е явен

1 Проблем?

```
public boolean priorTo (Time t)  {
        return  ( ( hour < t's hour) ||
            (hour == t's hour)&&(minute < t's minute)));
}
```

t не е получател, а аргумент

Вече използвахме "." нотацията за изпращане на съобщение към инстанция на даден клас.

"." нотацията се използва също от методите на един клас C за да видят променливи на инстанции на обекти от тип C, различни от получателя

МЕТОД PRIORTO

```
public boolean priorTo (Time t)  {
      return ((hour < t.hour) ||
            (hour == t.hour)&&(minute < t.minute)));
}
```

МЕТОД AFTER

- Искаме да разберем, дали получателят представя време, което е по-късно от t
- Една възможност: симетричен (обратен) метод на *priorTo*

1 Проблем?

Не съвсем коректен метод – при равно време връща true

```
public boolean after (Time t)  {
      return !priorTo(t);
}
```

НОВА ПОДОБРЕНА ВЕРСИЯ НА AFTER

1 Проблем?

Нямаме толкова лесно решение, както до сега – предаваме *receiver* като параметър

```
public boolean after (Time t)  {
      return t.priorTo(receiver-of-this-message);
}
```

2 Решение?

БЕЛЕЖКИ КЪМ МЕТОДА

- Отново затруднение, като при предишния метод – само, че тук нямаме толкова лесно решение
  - Проблемът е, че отново трябва да реферираме получателя на съобщението
- В предишния пример решението беше лесно, понеже можехме да изпратим съобщение до получателя, без да е необходимо да го именуваме явно
  - Тук не е така лесно, понеже предаваме получателя на актуалното съобщение като аргумент, а не изпращаме съобщение към него
- Java доставя начин за директно реферираne на получателя на едно съобщение.
  - В един метод на инстанция променливата **this** винаги оперира като референция на получателя на съобщението.
  - Следователно, **this** е декларирана неявно във всеки метод на инстанция от един клас

THIS

- Java предоставя възможност за отнасяне директно към получателя на съобщението
- В един метод на инстанция this винаги служи като референция към получателя на съобщение
- this винаги е декларирана неявно във всеки метод на инстанция на един клас

МЕТОД AFTER

```
public boolean after (Time t)   {
        return t.priorTo(this);
}
```

ПО-СЛОЖНА JAVA-ПРОГРАМА:TIMEPLAN

- Как намираме подалгоритми?
- Задача:
  - Искаме да управляваме нарастваща последователност от времена, чрез задаване на часа, следван от означение на започващо мероприятие
  - Със задаване на времето се резервира цялата, необходима за мероприятието продължителност, т.е. следващото свободно време
  - Актуалното време трябва да бъде представяно в съответствие с английската конвенция (AM: преди обяд до вкл. 12:00, PM: след обяд, *noon* за 12 ч., *midnight* за 0.00 ч.)
  - Завършващото време на плана трябва да бъде показано: време и кореспондиращата минута на деня.

РЕЗУЛТАТ НА ПРОГРАМАТА

3 Какви подалгоритми?

Общ алгоритъм

```
% java TimePlan
Calendar: Time + Text
---------------------------------
8:30AM            L Java
10:00AM           Break
10:15AM           L Software Engineering
11:45AM           Break
12:15PM           T Java
last (actual) Time in Minutes:
1:45PM = 825. Minute of day
```

ДАННИ И ПОДАЛГОРИТМИ

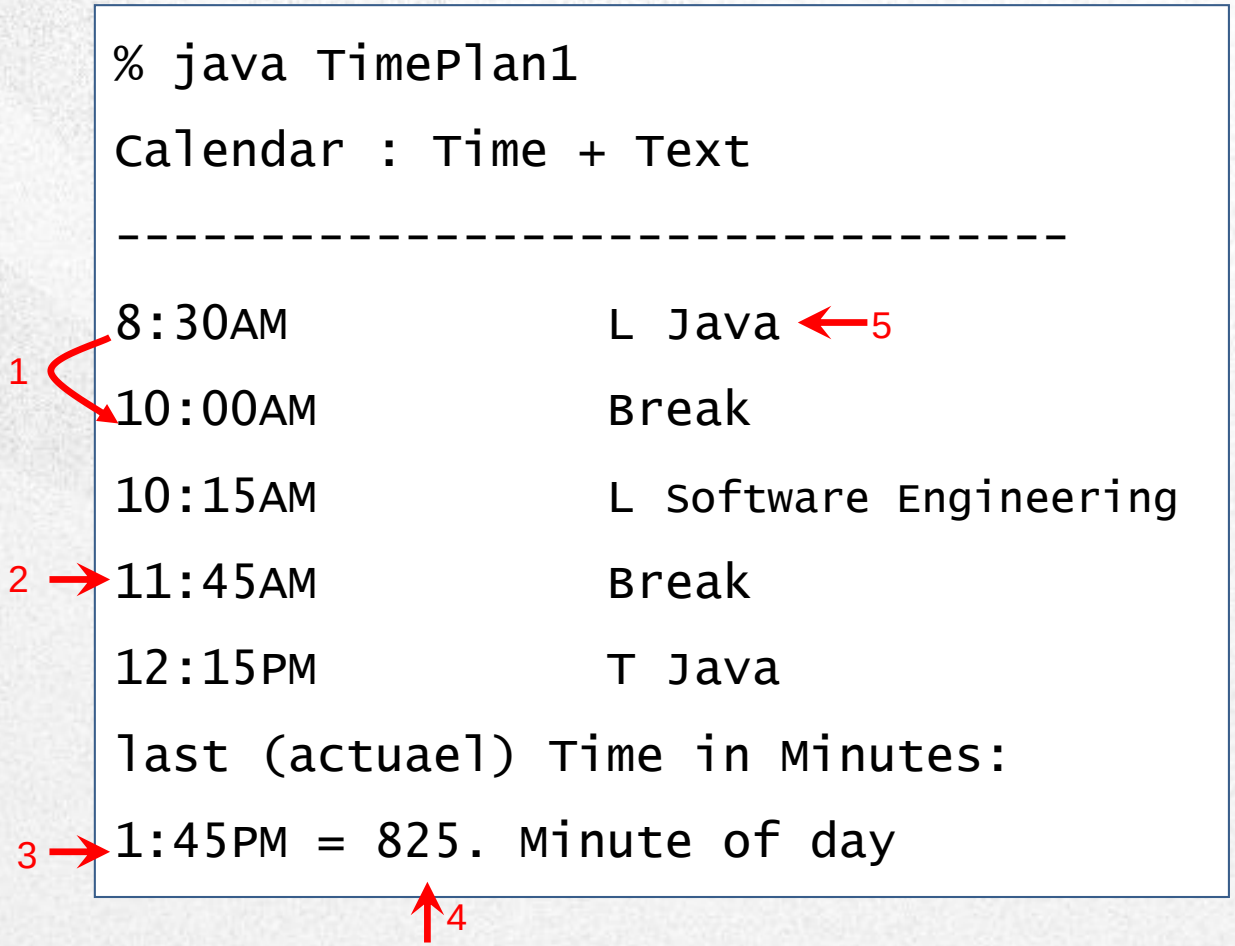
Идея:

→ Управление на актуалното време:
2 променливи (hour, minute)

→ Методи (алгоритми): за подзадачи

1. Увеличаване на актуалното време (addMinutes)
2. Показване на времето според конвенцията
3. Показване на времето в кореспондиращи брой минути
4. Преобразуване на времето в минути
5. Ново въвеждане (Време, Мероприятие)

ПОДЗАДАЧИ


% java TimePlan1
Calendar : Time + Text
8:30AM L Java
10:00AM Break
10:15AM L Software Engineering
11:45AM Break
12:15PM T Java
last (actuael) Time in Minutes:
1:45PM = 825. Minute of day
1
2
3
4
5

ИЗКУСТВОТО ИМПЕРАТИВНО ПРОГРАМИРАНЕ

Изкуството на императивното програмиране:

Намиране на подходящи подзадачи (алгоритми) и реализиране като процедури (методи, функции).

Така: ориентиране към смислени подзадачи, които имат принос за решаване на общата задача.

Внимание: оптимално декомпозиране обикновено не се получава при първия опит.

TIMEPLAN:ПРЕГЛЕД НА МЕТОДИТЕ

```
class TimePlan {
   private static int hour, minute;

   private static void addMinutes (int m)
   private static void printTime ()
   private static void printTimeInMinutes ()
   private static int timeInMinutes ()
   private static void includeNewEntry
               (int intervalInMinutes, String event)

   public static void main (String[] args) {
        ...
        includeNewEntry(90, „L Java");
        includeNewEntry(15, „Break");
        includeNewEntry(90, „L Software Engineering");
        ...
   }
}
```

ВРЕМЕТО, БРОЙ МИНУТИ

```
private static void printTimeInMinutes () {

     // показва времето със съответните минути

    printTime ();
    System.out.println("   = " + timeInMinutes()
                       + ". Minute of day ");
  }

private static int timeInMinutes () {

    // изчислява броя на минутите от 0:00,
    // които отговарят на актуалното време

    int totalMinutes = (60*hour + minute) % (24*60);
    if (totalMinutes < 0)
      totalMinutes = totalMinutes + 24*60;
    return totalMinutes;
}
```

НОВО ВЪВЕЖДАНЕ

```
private static void includeNewEntry
      (int intervalInMinutes, String event) {

   printTime();
   System.out.println("\t\t" + event);
   addMinutes(intervalInMinutes);
}
```

```
public static void main (String[] args) {
   hour = 8; minute = 30; //начало
   System.out.println("Calendar:...");
   System.out.println("-----------------");
   includeNewEntry(90, "L Java");
   includeNewEntry(15, "Break");
   includeNewEntry(90, "L Software Engineering");
   includeNewEntry(30, "Break");
   includeNewEntry(90, "T Java ");
   . . .
}
```

ПРОДЪЛЖИТЕЛНОСТ НА ПРОМЕНЛИВИТЕ

1. Какво е продължителност живот на променлива?

Продължителност на живот: Времеви интервал на съществуване на променливи по време на изпълнение на програмата (т.е.: памет)

2. Какви променливи?

Глобални променливи (static): общ run-time на програмата
- Обща ‘памет за данни‘ на повече методи

Локални променливи: от началото до края на извикване на метода (след това стойността се губи)
- Помощни променливи на алгоритъма
- Резервира се памет само по време на обработка на метода

ПРИМЕР

1 Какви променливи?

```
class TimePlan {
    private static int hour, minute;
          . . .
    private static void addMinutes (int m)    {
          int totalMinutes = . . . ;

          hour = ... ;
          minute = ... ;
    }
    public static void main (String[] args) {
          addMinutes(30);
    }
}
```

ВИДИМОСТ НА ИДЕНТИФИКАТОРИТЕ

- Вътре в класа:
  - **Локални променливи**: само в метода
  - **Глобални променливи**: в целия клас (възможно: скрити чрез локални променливи)
- Програмата се състои от повече класове:
  - **private-променливи/методи:** само в техния клас
  - **public-променливи/методи:** също в други класове

Видимост: Къде може да се използва едно име (EBNF: Identificator) на променлива в програмата?

ВИДИМОСТ НА ПРОМЕНЛИВИТЕ

- Вътре в класа:
  - **Локални променливи:** само в метода
  - **Глобални променливи:** в целия клас (възможно: скрити чрез локални променливи)

```
class Timeplan {
   private static int hour, minute;
      . . .
   private static void addMinutes (int m)    {
     int totalMinutes = . . . ;
     totalMinutes = ... ;
     minute = ... ;
   }
   public static void main (String[] args) {
     minute = 30;  // не totalMinutes = ...
   }
}
```

ВИДИМОСТ НА ПРОМЕНЛИВИТЕ И МЕТОДИТЕ

- Програмата се състои от повече класове:
  - private-променливи/методи: само в техния клас
  - public-променливи/методи: също в други класове

```
class Tumeplan {
   private static int hour, minute;
        . . .
   private static void addMinutes (int m)    {
        ...
   }
   public static void main (String[] args) {
        ...
   }
}
```

ОБОБЩЕНИЕ: НЯКОИ ОСНОВНИ ПРИНЦИПИ

- Декомпозиране на общата задача
  - На всяка алгоритмична подзадача се разработва кореспондиращ метод
  - В примера: 5 алгоритмични подзадачи → 5 метода
- Глобални променливи
  - Общи данни за повече методи
- Локални променливи
  - Помощни променливи на алгоритмите
- Концепции:
  - Продължителност на живот
  - Видимост
  - Формални и актуални параметри

УПРАЖНЕНИЯ С КЛАСА TIMEPLAN

- Обработка на програмата
- Разбиране на програмата
- Добавяне на нова функционалност:
  - Изваждане минути
  - Сравняване времена
  - Добавяне часове, ...

Изучаване програмирането посредством разглеждане и работа със съществени примерни програми

РАЗЛИКИ МЕЖДУ ДВАТА МЕТОДА

```
class Time {
   private int hour, minute;

   public Time (int h,int m) { hour = h; minute = m; }

   piblic Time addMinutes (int m) {
    int totalMinutes = (60*hour + minute + m)%(24*60);
    if (totalMinutes < 0)
         totalMinutes = totalMinutes + 24*60;
    return new Time(totalMinutes/60, totalMinutes%60);
   }

   public void printTime () { ... }

    public void advanceMinutes (int m) {
         int totalMinutes = (60*hour + minute + m)%(24*60);
         if (totalMinutes < 0)
              totalMinutes = totalMinutes + 24*60;
         hour = totalMinutes/60;
         minute = totalMinutes%60;
        }

  }
}
```

ПОСТОЯННИ И НЕПОСТОЯННИ КЛАСОВЕ

- Всички методи в първия метод на Time използват конструктури за присвояване стойности на обектните променливи
- Някои от методите връщат нови обекти в различни променливи
  - Такива класове се наричат ‘постоянен (неизменен)’ (immutable)
- Възможно е да използваме и други класове, наречени ‘непостоянен’ (mutable)
  - Променят обектните променливи на получателите и не връщат резултат

ПСЕВДОНИМИ (ALIASING)

- Две различни променливи могат да реферират един и същ обект
- Пример:

```
Time dawn,
     sunrise;

dawn = new Time(5,35);
sunrise = dawn;

....
```

БЛАГОДАРЯ ЗА ВНИМАНИЕТО!

КРАЙ “ВЪВЕДЕНИЕ В ООП”

Distributed @learning center