ВЪВЕДЕНИЕ

ЛЕКЦИОНЕН КУРС "ПРОГРАМИРАНЕ НА JAVA"

Distributed @learning center

СТРУКТУРА НА ЛЕКЦИЯТА

- Подобласти на информатиката
- Дейности на информатиците
- Защо Java?
- Бележки по лекционния курс

ПОДОБЛАСТИ НА ИНФОРМАТИКАТА

Практическа информатика:

- Бази данни
- Софуерно инженерство
- Езици за програмиране
- Компилатори
- Операционни системи
- Анализ на системи
- Дигитални медии
- ...

Теоретична информатика:

- Теория на алгоритмите
- Теория на автоматите
- Теория на изчислимостта
- Формални езици
- Теория на кодирането
- Теория на комплексността
- Логика
- Верификация на програми
- Теория на графите
- ...

ПОДОБЛАСТИ НА ИНФОРМАТИКАТА

Техническа информатика:

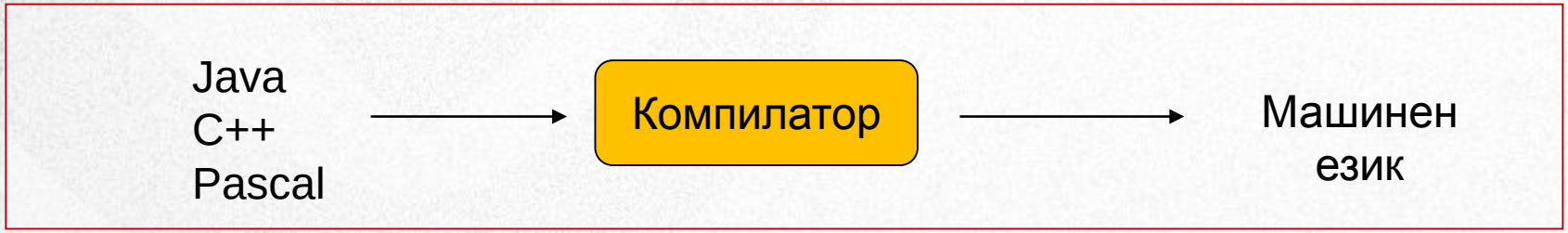
- Изчислителни структури
- Компютърни мрежи
- Обработка на сигнали
- ...

Приложна информатика:

- Бизнес-информатика
- Медицинска информатика
- Биоинформатика
- ...

СЪОТНОШЕНИЕ МЕЖДУ ПОДОБЛАСТИТЕ

1. Съществува ли ясно разделение?
2. Възможно ли е разработване на софтуер без теория?



Структура на първичен език :
• *синтаксис (граматики)*

Компоненти на компилатора :
• скенер: *крайни детерминирани автомати*
• парсер: *граматики*
• семантичен анализ: *атрибутирани граматики*
• генериране, оптимиране на код: *теория на графи*

Изчислителна архитектура :
• *структура на паметта*
• *машинни операции*

СЪЩЕСТВЕНИ ИЗВОДИ

- Разделянето на областите в обучението:
  - Основно по методически аргументи
- За практическа информатика е съществено:
  - Знания по теоретичните и техническите основи на информатиката са предпоставка за задълбочено разбиране процесите на разработване на софтуер

ЗА ПРИМЕРА

Практическа информатика:	***Теоретична информатика:***
- Бази данни - Софтуерно инженерство - Езици за програмиране - Компилатори - Операционни системи - Анализ на системи - Дигитални медии ...	- Теория на алгоритмите - Теория на автоматите - Теория на изчислимостта - Формални езици - Теория на кодирането - Теория на комплексността - Логика - Верификация на програми - Теория на графите ...
Техническа информатика:	***Приложна информатика:***
- Изчислителни структури - Компютърни мрежи - Обработка на сигнали ...	- Икономическа информатика - Медицинска информатика - Биоинформатика - ...

ФОРМАЛНИ ЕЗИЦИ И ГРАМАТИКИ

1 Защо са необходими формални граматики?

За да бъде точно дефинирано създаването на коректни програми

2 Кой проверява програмите са коректност?

```
class T1 {

   publick static main (...) {

   { x = 2 }
}
```

- Компилатор (К)
- Програмист (П)
- К & П

АСПЕКТИ НА КОРЕКТНОСТТА

Лексика:
Коректни символи?

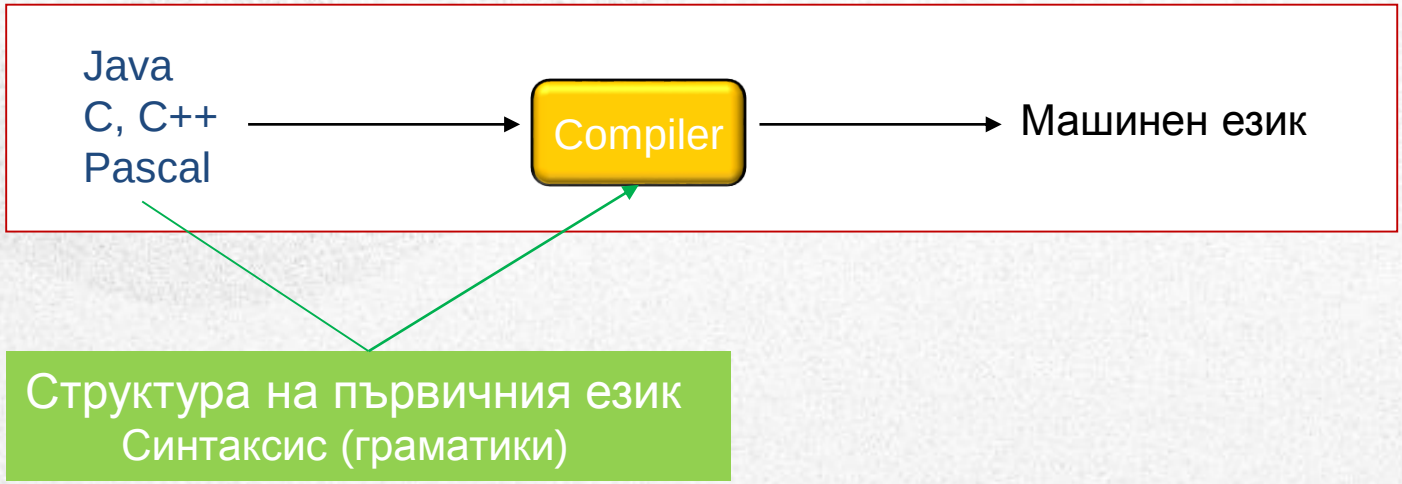
Контекстно-независим синтаксис:
Последователността на символите коректна?

```
class T1 {
    publick static main (...) {
    { x = 2 }
}
```

Контекстно-зависим синтаксис:
Символите коректно свързани в околната среда?

Семантика:
Грешки при изпълнението ?
Обработката на програмата коректна?

АНАЛИЗ НА ГРЕШКИТЕ ОТ КОМПОНЕНТИТЕ НА КОМПИЛАТОРА



ДЕФИНИЦИЯ НА СИНТАКСИСА НА ЕЗИЦИТЕ ЗА ПРОГРАМИРАНЕ

- Контекстно-свободен синтаксис с контексно-свободни граматики (EBNF)
  - Напълно формализирани
- Контекстно-зависим синтаксис
  - Вербално задаван
  - Напр. , всеки идентификатор трябва да бъде деклариран преди неговото използване

ЗАДАЧА НА КОНТЕКСТНО-СВОБОДНИТЕ ГРАМАТИКИ

1 Каква е задачата на контекстно-свободните граматики?

Различава коректните от грешните програми

```
class c { int x = 1; }
```

```
class { int x:= 1 }
```

АЗБУКА

Крайно непразно множество A от символи $a \in A$

напр. основни символи на един език за програмиране:

A_{Java} = { **`class`**, **`public`**, **`{`**, **`}`**, **`<`**, **`>=`**, **`==`**, **`=`**, **`...`** }
A_{Pascal} = { **`program`**, **`procedure`**, **`{`**, **`}`**, **`<`**, **`>=`**, **`=`**, **`:=`**, **`...`** }

Основни символи
- знаци
- последователности от знаци
  - единични знаци: <
  - двойни знаци: <= >=
  - по-дълги: низове

ДУМИ ВЪРХУ АЗБУКАТА

Множество на думите A^* върху една азбука A (символни низове)

Индуктивна дефиниция:

- Празното множесто ε принадлежи към A^* (празен символен низ)
- Конкатенацията на един символен низ $x \in A^*$
  с един символ $a \in A$ $(x \otimes a)$ е отново символен низ $xa \in A^*$
- Други елементи от A^* не съществуват

КОНТЕКСТНО-СВОБОДНИ ГРАМАТИКИ

$G = [A,M,s,R]$ се нарича контекстно-свободна граматика, ако:

- A азбука: основни символи, терминални символи, терминали
- M азбука: (метасимволи, нетерминали)
  $A \cap M = \varnothing$ (дизюнктивни множества)
- $s \in M$: стартов символ
- R крайно множесто на правила: (синтактични, продуктивни, заместващи)
  $R \subseteq M \times (A \cup M)^*$, т.е. множество от елементи (l, r) с $l \in M$, $r \in (A \cup M)^*$

ПРИМЕРИ : ДЕКАРТОВО ПРОИЗВЕДЕНИЕ

1 Колко елемента?

1. A = {1, 2, 3}
   B = {x, y}

   A x B = { (1,x), (2,x), (3,x), (1,y), ...} 6

2. N = {0, 1, 2, 3, …} (естествени числа)
   L = {a, b, c, …} (букви)

   N x L = { (0,a), (0,b), ... , (0,z), (1,a), ... } ∞

3. Общо за граматиките:

   M – метасимволи
   A – терминални символи

   M x (A ∪M)* крайни или безкрайни?

ПРИМЕР: ГРАМАТИКА ЗА ДЕФИНИЦИЯ НА ПРОСТИ ИЗРАЗИ

напр. a, a + a, a * a, a + a + (a * a + a) и т.н.

$G_1 = [A_1, M_1, s_1, R_1]$

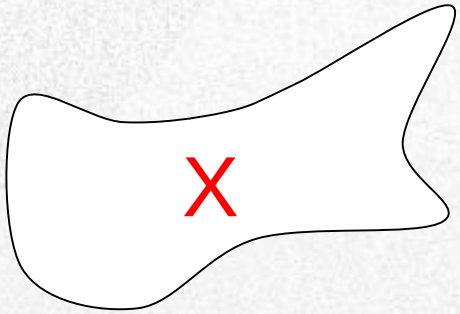
A_1 = { a, +, *, (,) }
M_1 = { expr, exprrest, term, termrest, factor }
s_1 = expr

R_1 с 8 правила:

((expr, term exprrest),
(exprrest,),
(termrest, * factor termrest),
(factor, a),
(exprrest, + term exprrest),
(term, factor termrest),
(termrest,),
(factor, (expr))

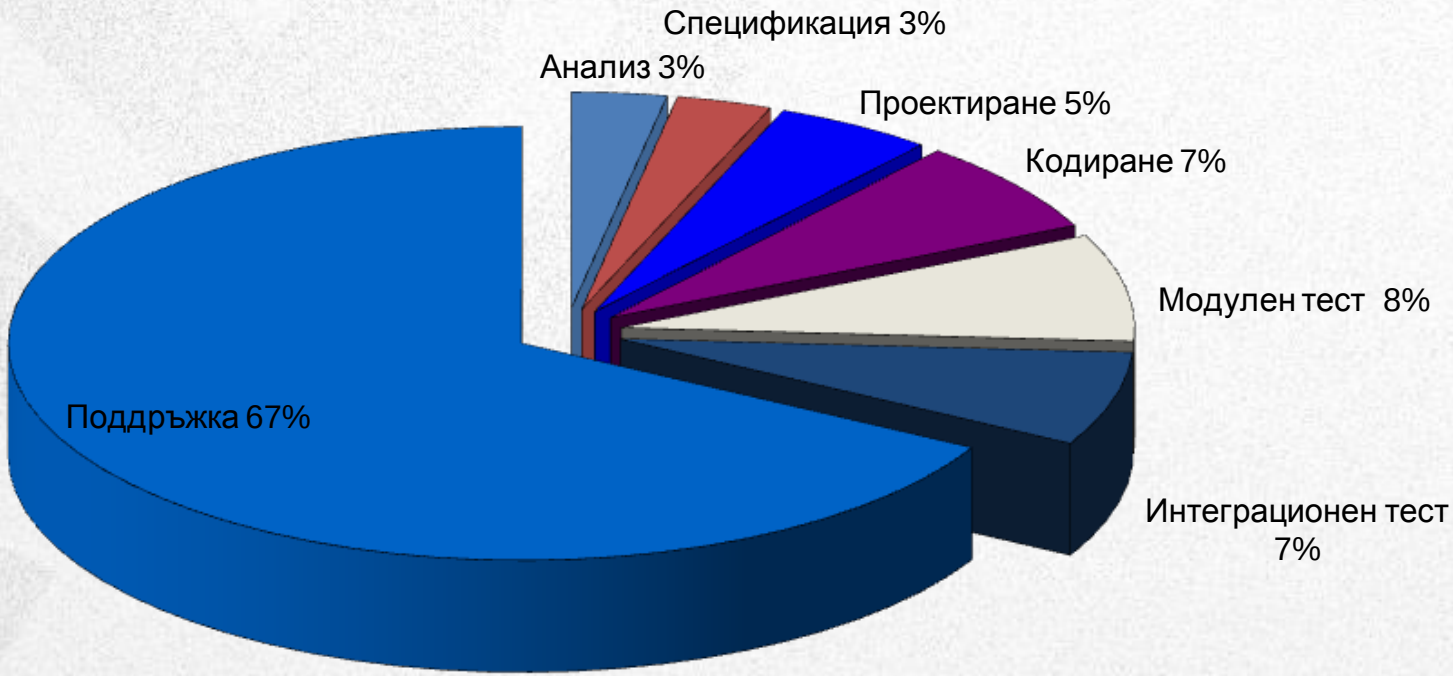
ДЕЙНОСТИ НА ИНФОРМАТИЦИТЕ

- Какво е съотношението на програмирането към другите дейности при разработване на една система?



X = програмиране ?

РАЗПРЕДЕЛЕНИЕ НА РАЗХОДИТЕ В ЖИЗНЕНИЯ ЦИКЪЛ НА СОФТУЕРА

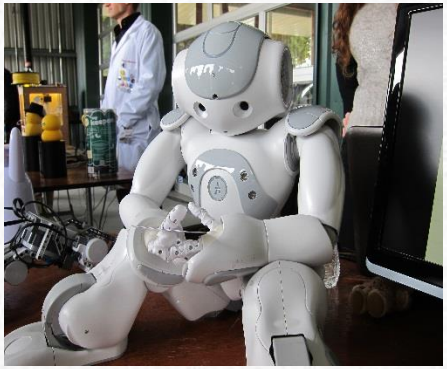


ЗАЩО JAVA?

- Опростен
- Обектно-ориентиран
- Преносим (машинно-независим)
- Удобен за Internet/WWW програмиране
- Възникване:
  - 1995
  - Sun
- Актуално
  - Развива се от Oracle

ИНТЕРЕСНИ ПРИЛОЖНИ ОБЛАСТИ

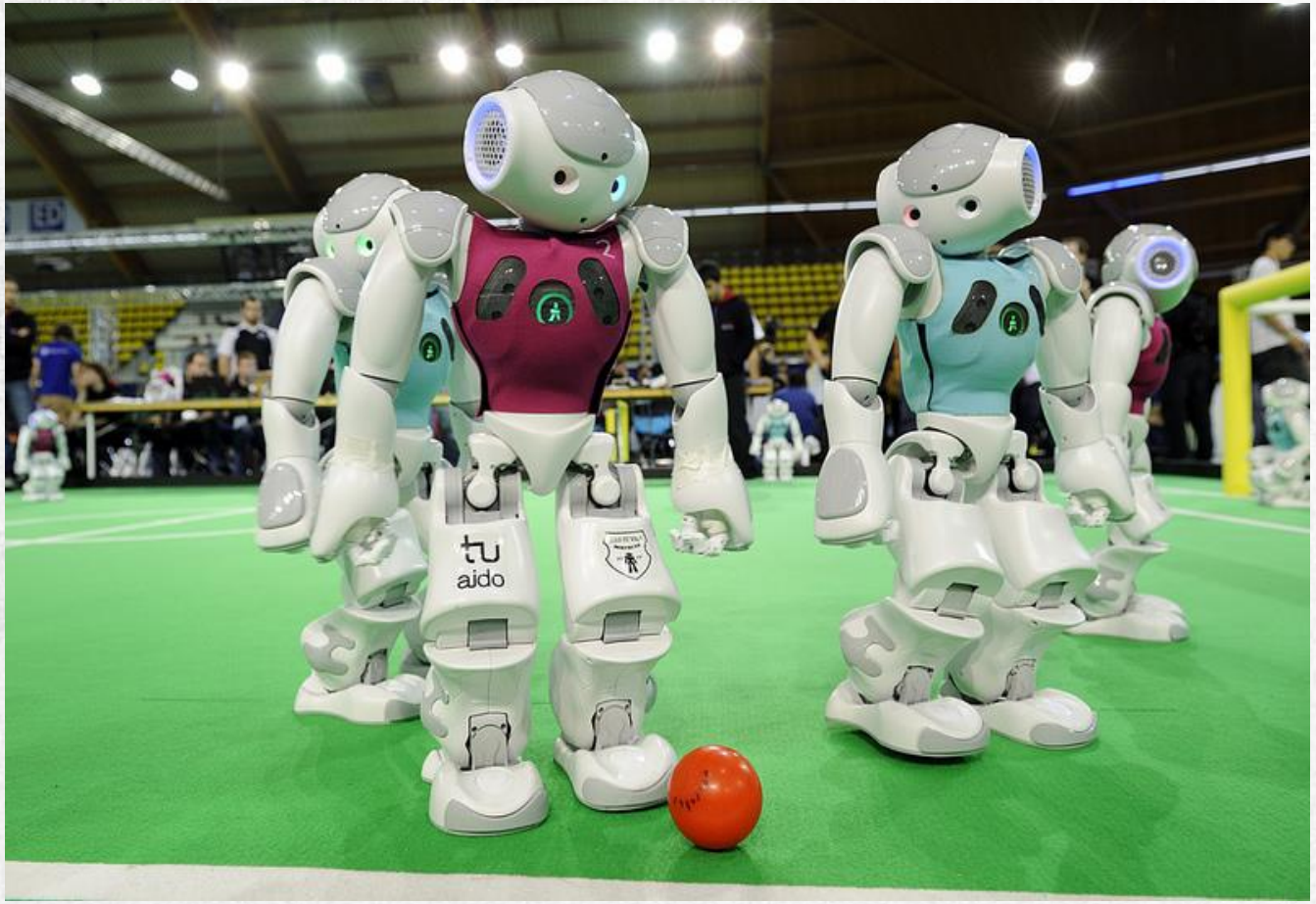
- Интернет
- Интелигентни системи
  - Интернет на нещата (Internet of Things)
  - Семантичен уеб (Semantic Web)
  - Персонални асистенти
- Мобилни устройства
  - Android
- Web услуги
- Роботика
- Медицинска апаратура
- Суперкомпютри
- Космическа апаратура
- …



GROVER

- NASA започна изпитанията на своя нов полярен изследователски робот GROVER (в Гренландия - тестове за издръжливост на ниски температури и силен вятър). Роботът се захранва от слънчеви батерии (осигуряват 12 часа автономна работа). Роботът притежава GPS навигация и мощен радар, който му позволява да анализира дебелината на снежната и ледената покривка. Команди за действия GROVER получава от САЩ чрез спътник. Роботът е способен сам да изпрати на изследователите събраната информация. Първоначално проектът GROVER е разработен в студентски научен екип (Годард - САЩ през 2010-2011 год.). След това NASA поема усъвършенстването и развитието на робота с намерение да го използва при изследване на други планети.

ROBOCUP



ТЕМИ В ЛЕКЦИОННИЯ КУРС

Основи

- Граматики
- Представяне на данни
- Фон-Нойман-компютри
- Езици за програмиране: класификация
- ...

Структури данни и алгоритми

- Списъци
- Дървета
- Сортиране и търсене
- Комплексност на алгоритми O(n)
- ...

ТЕМИ В ЛЕКЦИОННИЯ КУРС

Концепции на (императивните) езици за програмиране

- Променливи: видимост, жизнен цикъл,
- Типове данни (прости, структурирани)
- Изрази: приоритети, cast...
- Оператори (прости, структурирани)
- Методи като абстракция
- Параметри: value –ref
- Рекурсия – итерация
- Сравнение на езиците + критика
- ...

ОО концепции

- АТД - обекти - класове,
- Наследяване
- Видимост
- Променливи и методи на класове
- Абстрактни класове
- Претоварване
- Полиморфия
- Динамично свързване
- Обработка на изключения
- Събития
- API (избрани класове)
- Аплети

ТЕМИ В ЛЕКЦИОННИЯ КУРС

Разработване на софтуер

- Качество на софтуера,
- Фази и документи,
- Абстракция + декомпозиция
- Архитектура: UML
- Видове компоненти
- Комплексни примери
- ...

Програмни умения

- Типични примери - еталони + познати примери: Ханой, стек, Quicksort, ...
- Упражнения
- Самостоятелна работа
- ...

ЗА ЛЕКЦИОННИЯ КУРС

- Съвместен лекционен курс:
  - Prof. Klaus Bothe, Humboldt University - Berlin
- Хорариум:
  - 30 часа лекции
  - 30 часа упражнения
- Съществено:
  - Самостоятелна работа

ЦЕЛИ НА ЛЕКЦИОННИЯ КУРС

- Изучаване на (втори) ОО език за програмиране: Java
- Като: Разширяваме и систематизираме наличните знания
- Предполагаме:
  - Различна предварителна подготовка
  - Евентуално понятийни различия (от предишни лекции, практикуми, курсови проекти, самообучение)
- Неизбежно:
  - Активно участие в упражненията
  - Самостоятелна работа – проекти, литература

МЕТОДИКА

- Обучение за създаване на Java програми чрез:
  - Добра теоретична подготовка
  - Разучаване на готови програми (learning by reading)
  - Личен практически опит в програмиране (learning by doing)
  - Изпълнение на малки проекти
- В рамките на този лекционен курс основен акцент:
  - Индивидуална подготовка и развитие на индивидуални способности

ОЦЕНЯВАНЕ

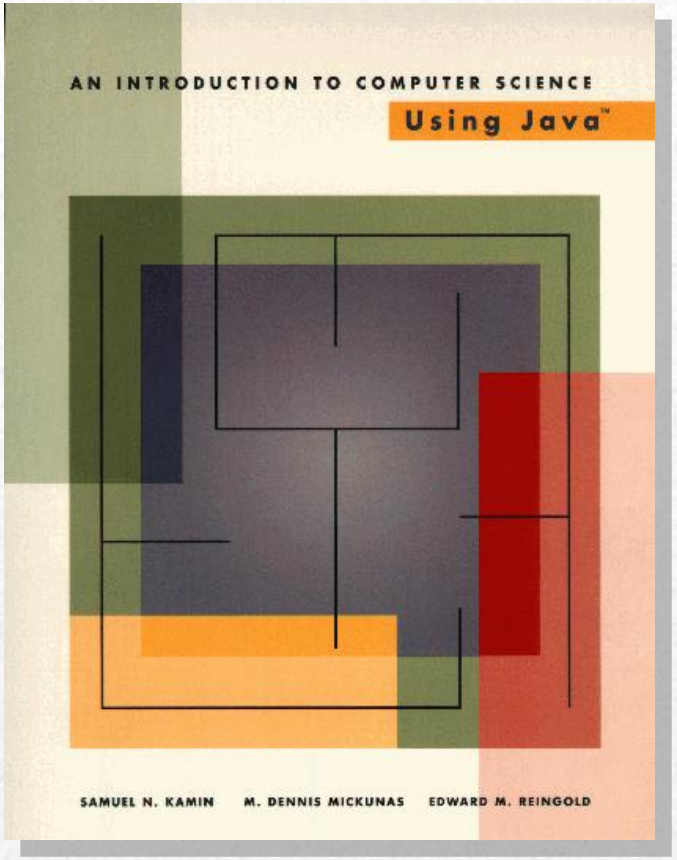
- Обхващане на всички елементи на образователния процес:
  - Теоретичен изпит (тест) – 0.50
  - Участие и работа в упражненията – 0.50
- Особено ще се стимулира:
  - Демонстрация на добри практически умения
  - Напр. чрез разработване на самостоятелни проекти

СТРУКТУРА НА ЛЕКЦИОННИЯ КУРС

- Въведение
- **Част I:** Императивни езикови концепции и структури
- **Част II:** Основи на обектно-ориентираното програмиране и разработване на софтуер в Java

ПРЕПОРЪЧИТЕЛНА ЛИТЕРАТУРА

1. S.N. Kamin, M.D. Mickunas, E.M. Reingold: „An introduction to computer science – Using Java“, McGraw-Hill, 1998
2. B. Eckel, Thinking in Java, Prentice Hall, 1998 and next editions
3. C.Horstmann, Big Java, John Wiley & Sons, 2002
4. J. Nino, F.A. Hosch, An Introduction to Programming and Object-Oriented Design Using Java, John Wiley & Sons, 2002



БЛАГОДАРЯ ЗА ВНИМАНИЕТО!

КРАЙ “ВЪВЕДЕНИЕ”

Distributed
@learning center