

Объектно-ориентированное программирование

Лекция 1: парадигмы программирования и
введение в Java

Об этом курсе

- Совместный курс для двух направлений двух факультетов
- 15 лекций по ООП, Java, JVM

05.09	Парадигмы программирования и введение в Java	
12.09	Понятия объекта и класса	
19.09	Основные принципы объектного подхода	
26.09	Наследование	
03.10	Механизм исключений и его реализация в JVM	
10.10	Понятие контракта, интерфейсы	
17.10	Таблица виртуальных методов	
24.10	Параметризованные типы	
31.10	Строки и коллекции, часть 1	
07.11	Строки и коллекции, часть 2	
14.11	Stream API	
21.11	Ввод и вывод	
28.11	JVM байткод	
05.12	Автоматическая сборка мусора	
12.12	JIT-компиляция	

О себе

- Анастасия Александровна Шадрина
- Бакалавр ФИТ НГУ, ассистент кафедры Систем информатики
- Область проф. интересов: статический анализ исходного кода
- Разработчик команды Excelsior в компании Huawei
- Более 5 лет опыта промышленной разработки на Java

Учебные ресурсы и где их искать

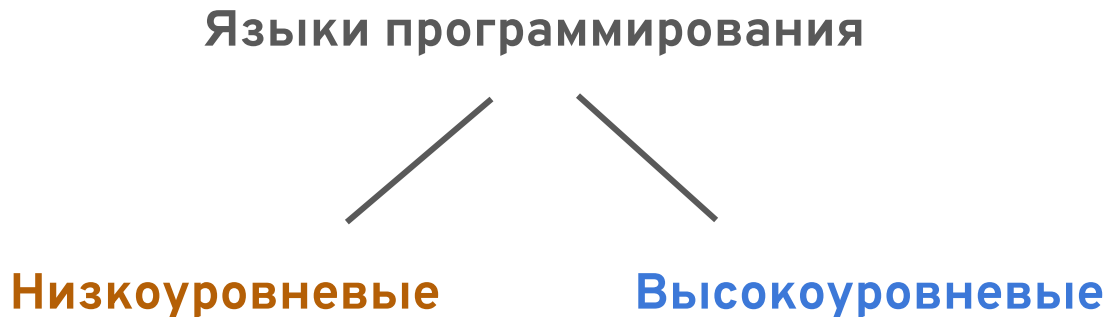
- Учебные материалы и объявления искать в [Google Classroom](#)
- Если у вас нет доступа, напишите на почту Александру Александровичу a.vlasov7@g.nsu.ru

Сегодня про

- Парадигмы программирования
- История появления Java
- Почему Java
 - Кроссплатформенность и как она достигается
 - Подробнее о JVM
 - Автоматическая сборка мусора
- Hello world!
- Пару слов о системах сборки Java приложений

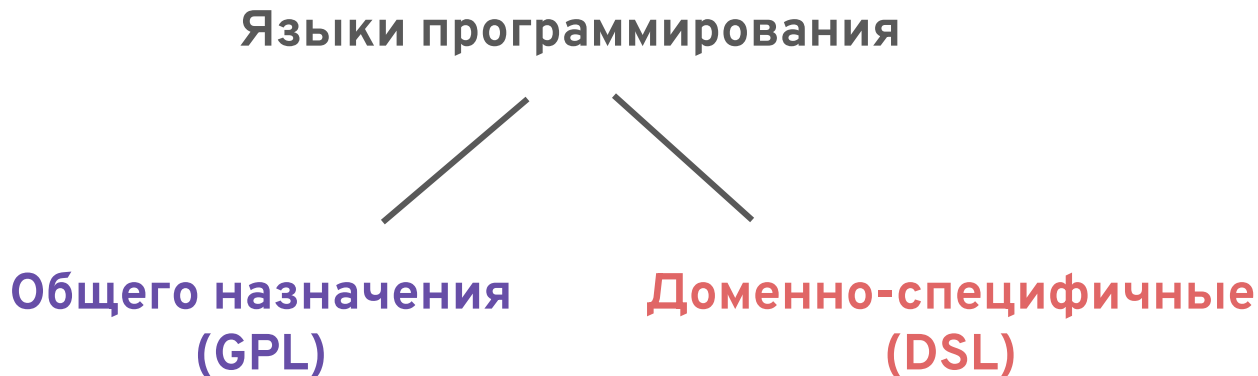
Про языки программирования в общем

Язык программирования (ЯП) – что-то между человеко-читаемым представлением и машинным кодом.



Про языки программирования в общем

Будем говорить про языки **высокого уровня общего назначения**.



Тенденции развития информационных систем

- Рост вычислительных ресурсов и объема информации
- Усложнение программных систем
- Усложнение инструментария разработки
- Широкий спектр специалистов разных профессий участвуют в разработке ПО
- Изменение требований к ПО
- Изменяющееся окружение исполнения

Разделение на парадигмы

Парадигма программирования – совокупность принципов, методов и понятий, определяющих способ конструирования программ.

Разделение на парадигмы

1. **Императивное (процедурное) программирование**
Последовательное выполнение инструкций с изменением состояния
2. **Структурное программирование**
Иерархическая структура блоков из ветвлений, циклов, групп операторов и подпрограмм
3. **Функциональное программирование**
Нет состояния, программа состоит из вычисления значения и композиции функций
4. **Логическое программирование**
Логический вывод на основе заданных фактов и правил вывода
5. **Объектно-ориентированное программирование**
Взаимодействие объектов

Императивное (процедурное) программирование

- Теоретическая модель – машина Тьюринга
- Память + последовательное выполнение команд

+ Просто для понимания

— Трудно масштабировать

Структурное программирование

- “Качество программного кода обратно пропорционально количеству операторов `goto` в нём”
- Без `goto` вся программа = иерархическая структура из ветвлений, циклов и блоков операторов
- Разделение на подпрограммы

✚ Пошаговая разработка, легко масштабировать, доказывать корректность, отлаживать и тестировать

— Не все задачи можно алгоритмически декомпозировать и даже описать

Функциональное программирование

- Нет состояния
- Программа = вычисление значения функции
- Не как делать, а **что** делать

✚ Нет побочных эффектов => надежность, доказательство корректности, просто тестировать и отлаживать, параллелить, оптимизировать

— На одних чистых функциях далеко не уедешь

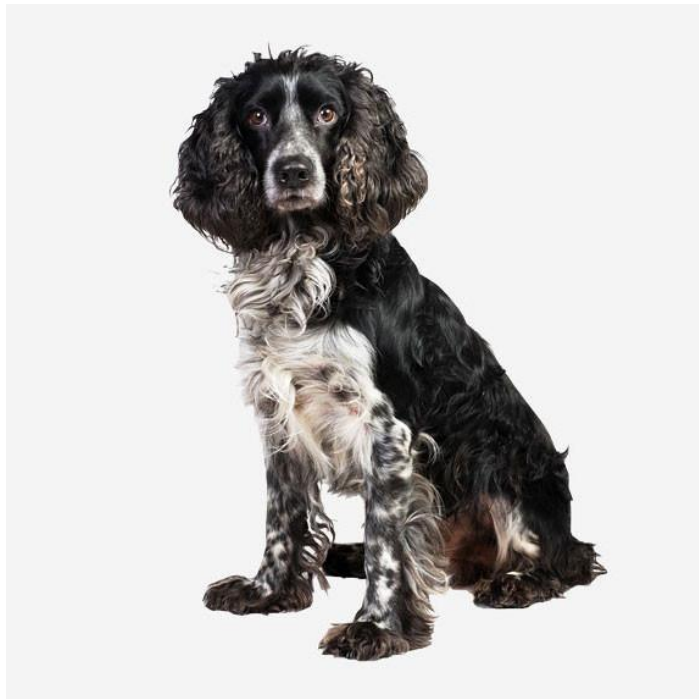
Логическое программирование

- Нет состояния
- Не как делать, а **что** делать
- Вместо композиции функций правила вывода логических утверждений

✚ Нет побочных эффектов => надежность, доказательство корректности, просто тестировать и отлаживать, параллелить, оптимизировать

— Подходит для определенного класса задач

Объект



- **Состояние**

Порода: русский спаниель, возраст: один год, окрас: черно-белый, характер: игривый

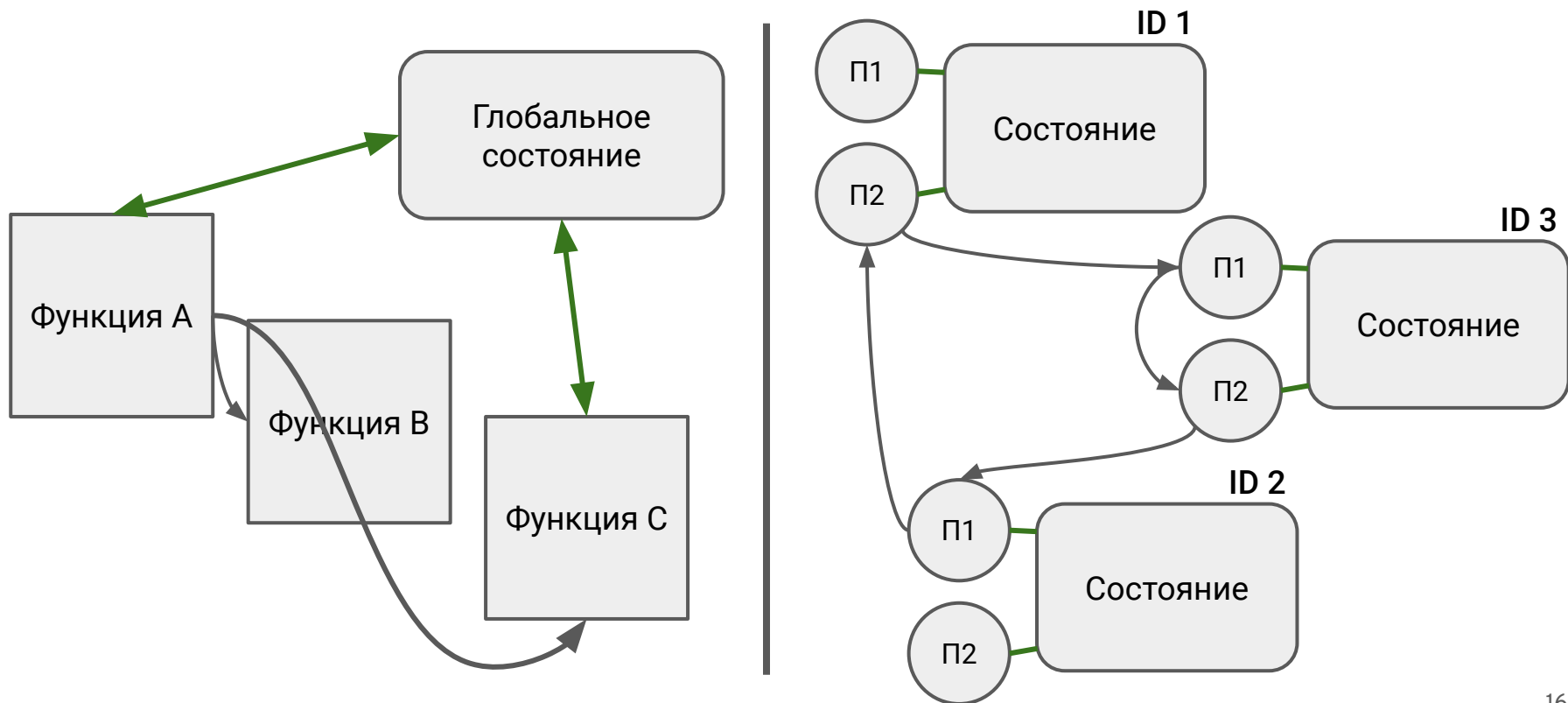
- **Поведение**

Спит, ест, реагирует на других собак

- **Идентичность**

Один единственный неповторимый

Объектно-ориентированный подход



Simula (1967)

- 1 -

CHAPTER 1.

SIMULATION AND DISCRETE EVENT SYSTEMS.

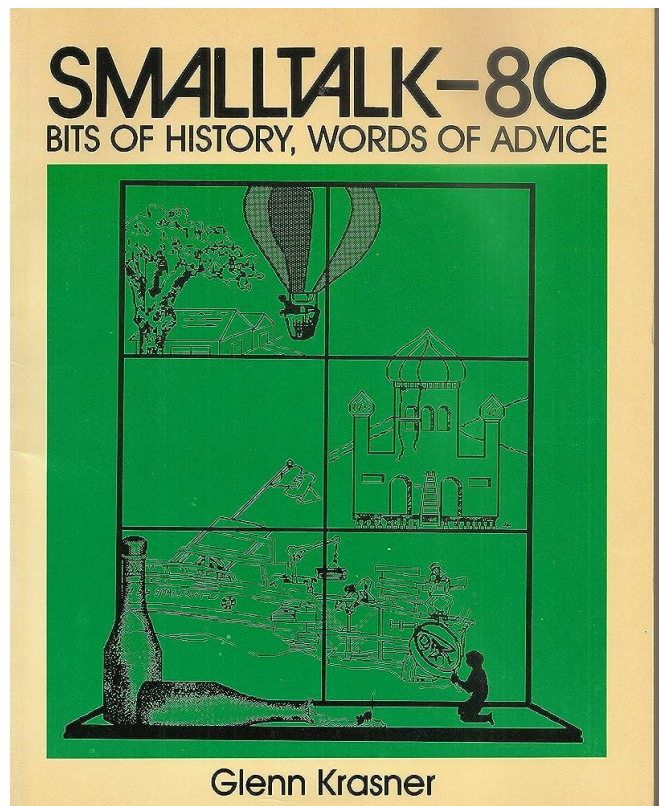
1.1 The SIMULA Project.

The two main objects of the SIMULA language are :

1. to provide a language for a precise and standardized description of a wide class of phenomena, belonging to what we may call "discrete event systems".
2. to provide a programming language for an easy generation of simulation programs for "discrete event systems".

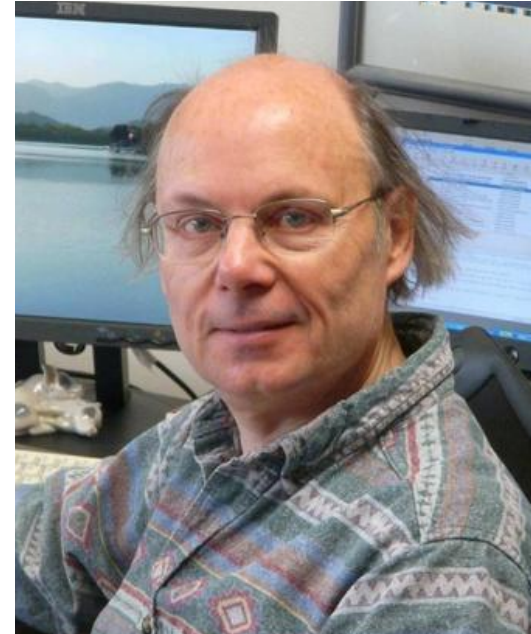


Smalltalk (1980)



C++ (1983)

C makes it easy to shoot yourself in the foot;
C++ makes it harder, but when you do it
blows your whole leg off.



Бьерн Страуструп

Предпосылки к появлению Java

- Рост разнообразия исполняемых устройств
- Платформенная зависимость программного кода
- Рост сложности программных систем
- Подверженность кода ошибкам
- Рост уровня необходимой квалификации разработчика

Oak (1991)

В 1991 г. появился Oak – язык для программирования бытовых устройств.

- Разрабатывался как альтернатива C/C++
- Платформенная независимость благодаря наличию виртуальной машины
- Сборка мусора



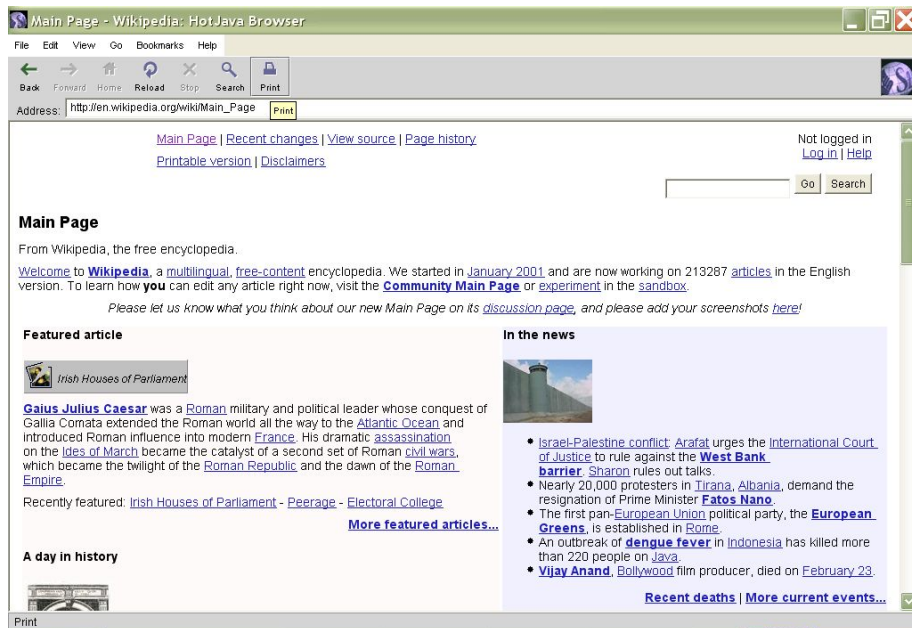
Джеймс Гослинг

Oak становится Java

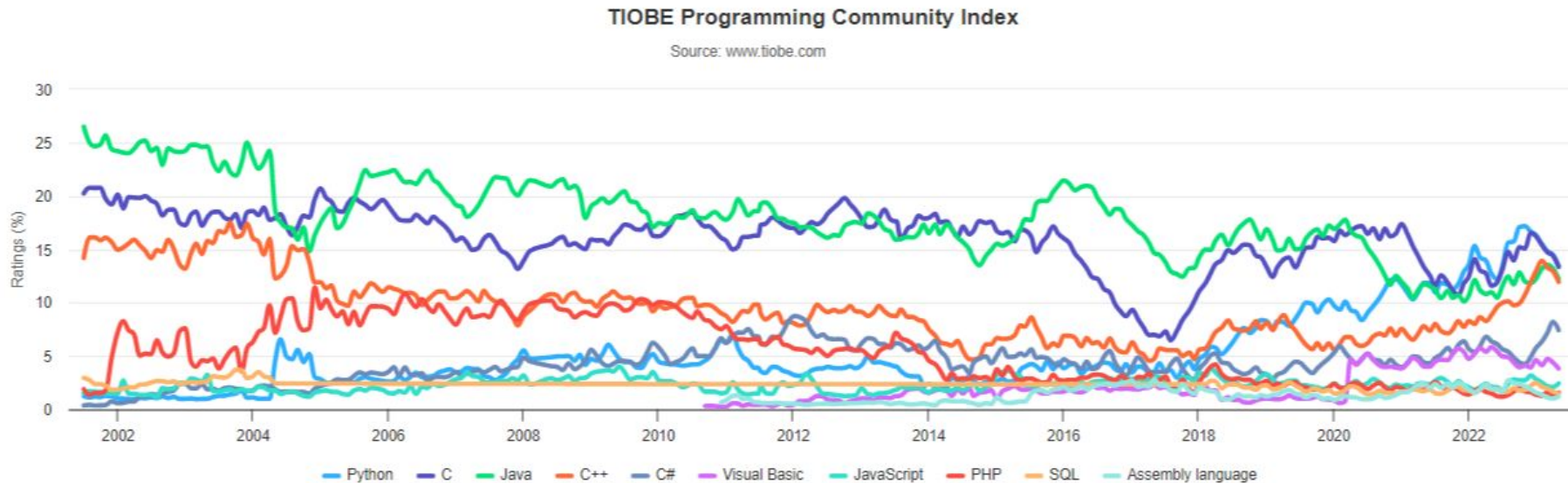
- Сентябрь 1992 г. – ручное устройство дистанционного управления Star7.
- Март 1993 г. – успешные результаты испытаний кабельного интерактивного телевидения.
- Апрель 1993 г. – выпуск первого графического браузера, разработанного в центре NCSA.
- Декабрь 1993 г. – отсутствие прогресса на рынке кабельного телевидения.
- Сентябрь 1994 г. – начало разработки браузера WebRunner.

Оак становится Java (1995)

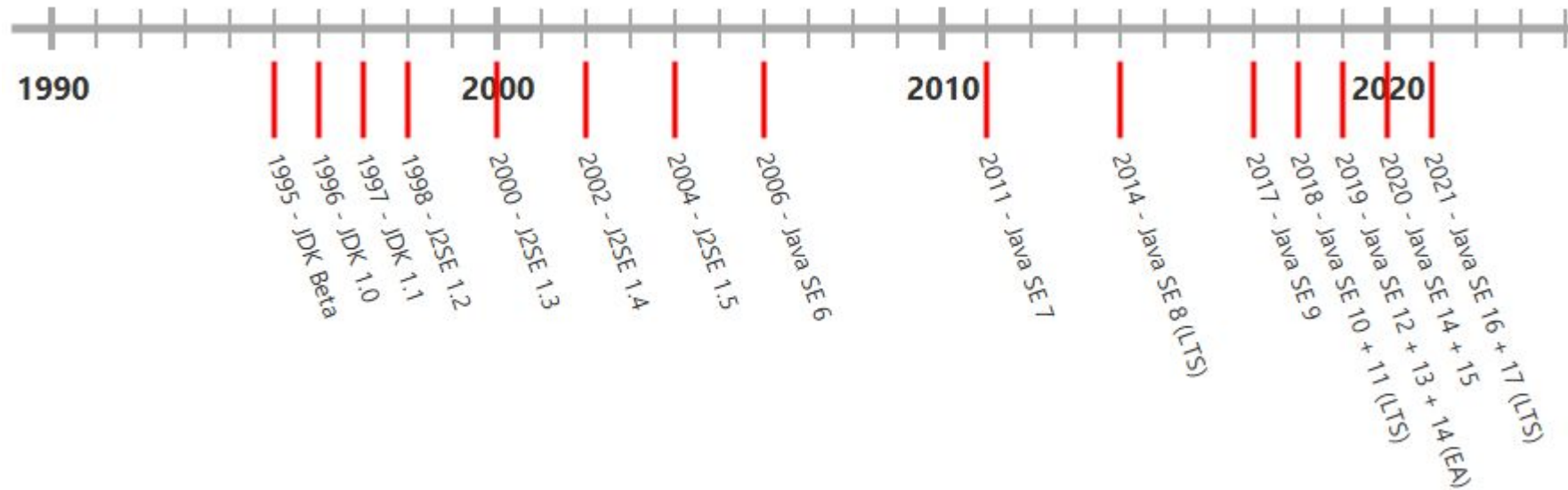
23 мая 1995 г. – компания Sun официально представляет язык программирования Java и браузер HotJava на выставке SunWorld '95.



Популярность Java



Эволюция Java



Принципы Java

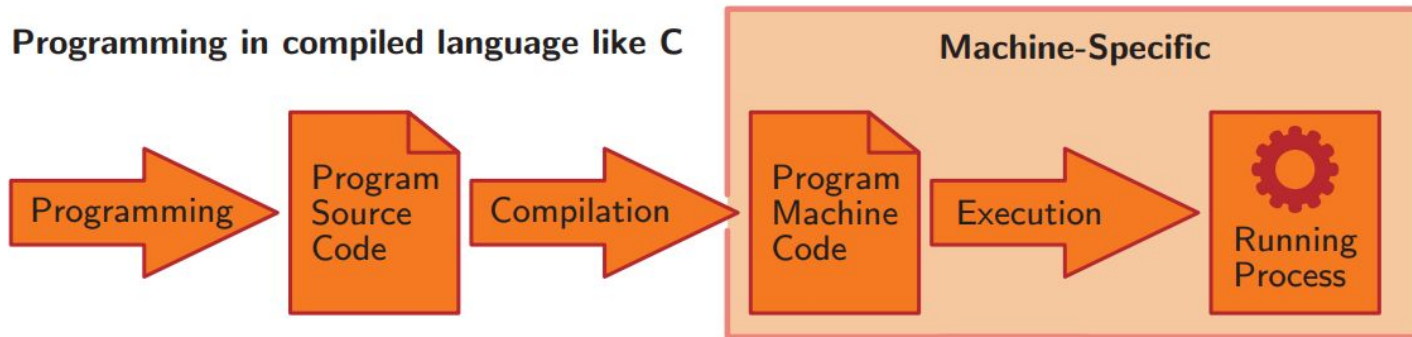
- Простой понятный синтаксис C
- Поддержка ООП
- Кроссплатформенность
- Производительность
- Автоматическая сборка мусора
- Отсутствие низкоуровневого управления памятью
- Поддержка многопоточности

Принципы Java

- Простой понятный синтаксис C
- Поддержка ООП
- Кроссплатформенность
- Производительность
- Автоматическая сборка мусора
- Отсутствие низкоуровневого управления памятью
- Поддержка многопоточности

Программирование на C

- Написание исходного кода
- Компиляция в объектные файлы
- Линковка в исполняемый файл
- Исполнение: программа загружается в память вычислительного устройства, машинный код выполняется процессором

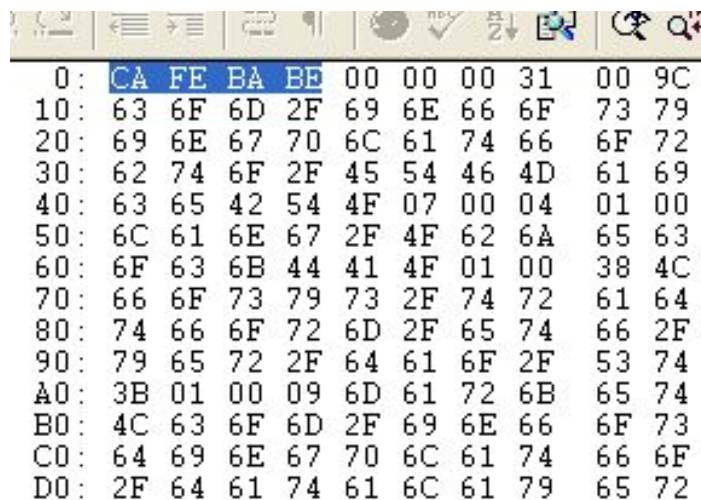


Программирование на Java

- Написание исходного кода

Программирование на Java

- Написание исходного кода
- Компиляция в **.class файлы (байткод)**

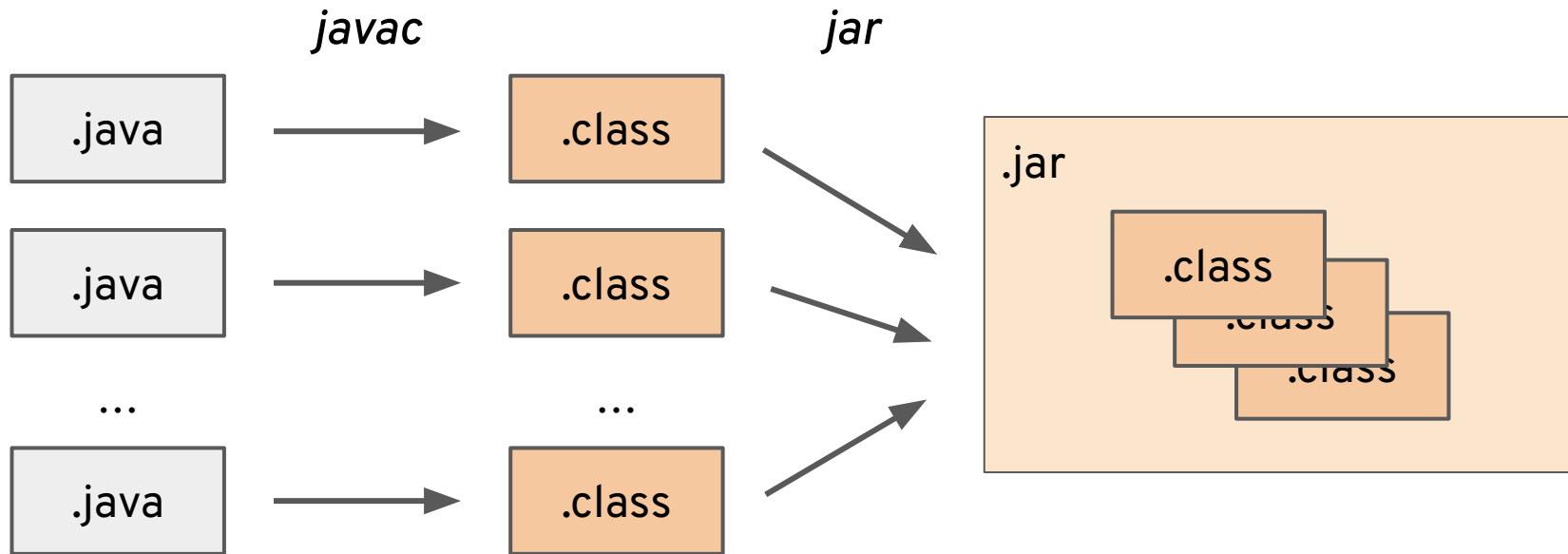


A screenshot of a hex editor window showing a memory dump. The interface includes a toolbar at the top with various icons for file operations, navigation, and editing. The main area displays a list of memory addresses and their corresponding hexadecimal values. The first four bytes (00000031) are highlighted in blue.

0:	CA	FE	BA	BE	00	00	00	31	00	9C
10:	63	6F	6D	2F	69	6E	66	6F	73	79
20:	69	6E	67	70	6C	61	74	66	6F	72
30:	62	74	6F	2F	45	54	46	4D	61	69
40:	63	65	42	54	4F	07	00	04	01	00
50:	6C	61	6E	67	2F	4F	62	6A	65	63
60:	6F	63	6B	44	41	4F	01	00	38	4C
70:	66	6F	73	79	73	2F	74	72	61	64
80:	74	66	6F	72	6D	2F	65	74	66	2F
90:	79	65	72	2F	64	61	6F	2F	53	74
A0:	3B	01	00	09	6D	61	72	6B	65	74
B0:	4C	63	6F	6D	2F	69	6E	66	6F	73
C0:	64	69	6E	67	70	6C	61	74	66	6F
D0:	2F	64	61	74	61	6C	61	79	65	72

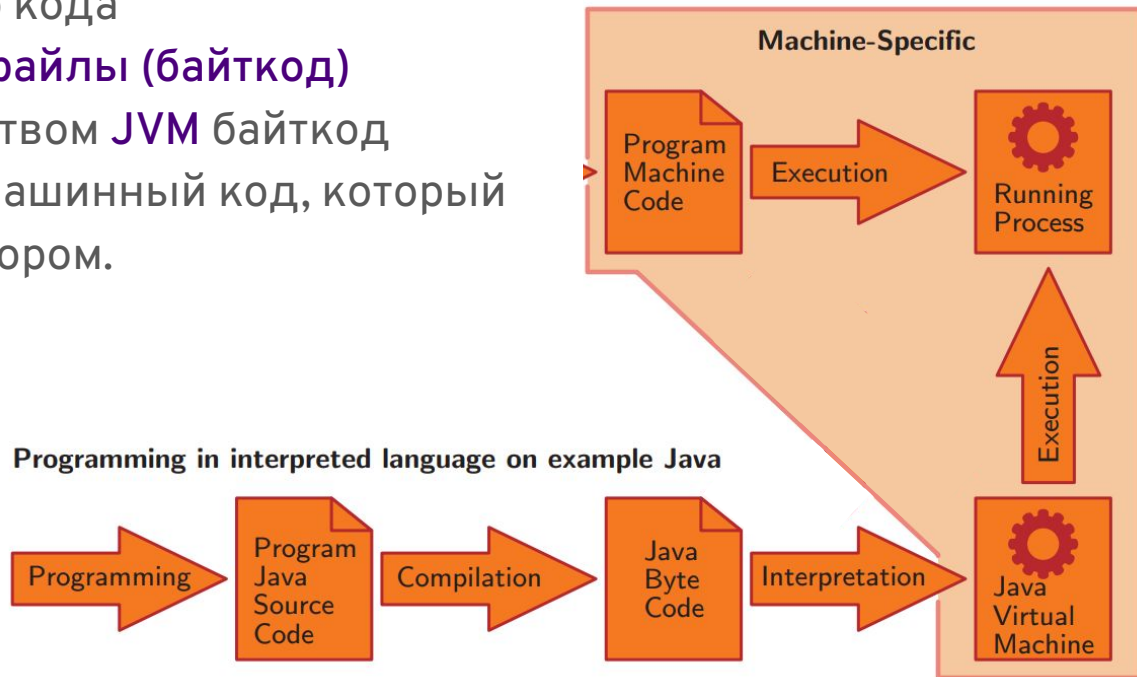
Программирование на Java

- Написание исходного кода
- Компиляция в **.class файлы (байткод)**



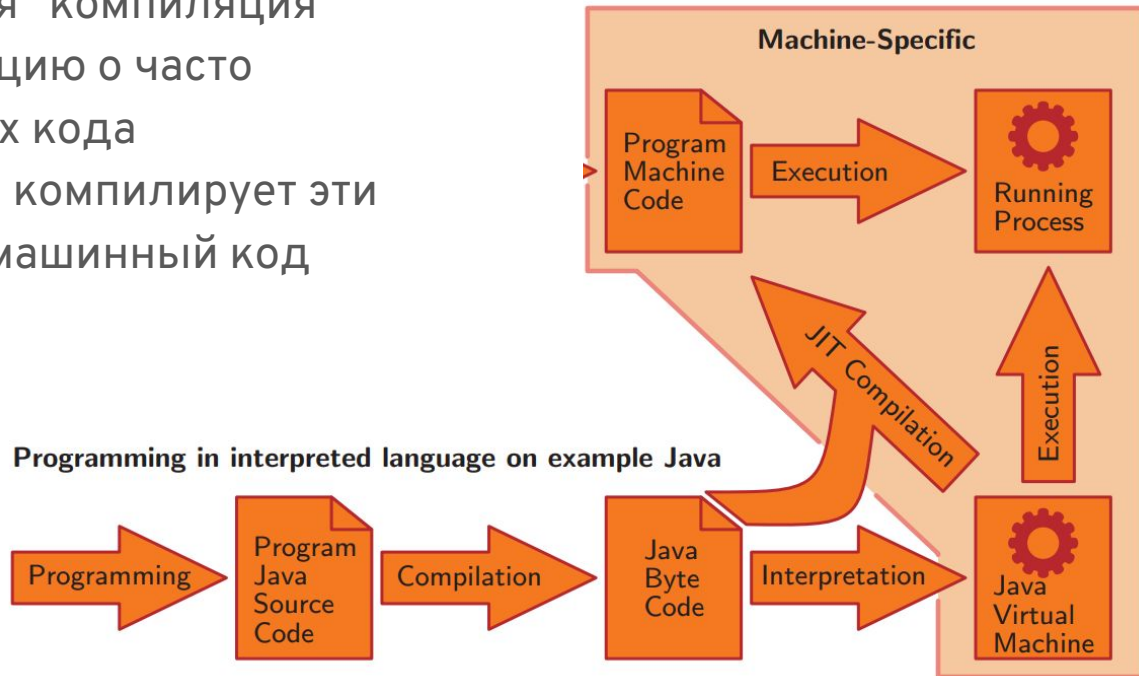
Программирование на Java

- Написание исходного кода
- Компиляция в **.class файлы (байткод)**
- Исполнение: посредством **JVM** байткод интерпретируется в машинный код, который выполняется процессором.



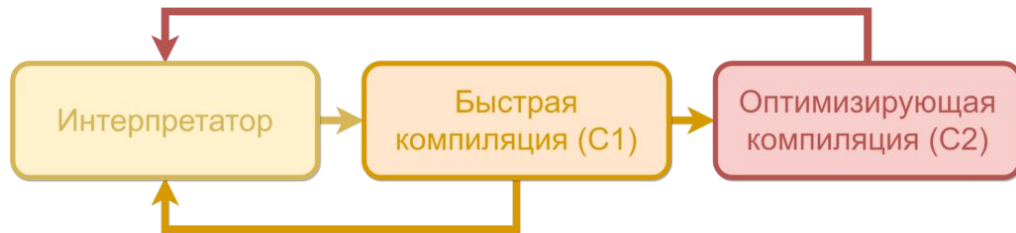
JIT компиляция

- Динамическая “умная” компиляция
- Использует информацию о часто вызываемых участках кода
- Во время исполнения компилирует эти участки напрямую в машинный код



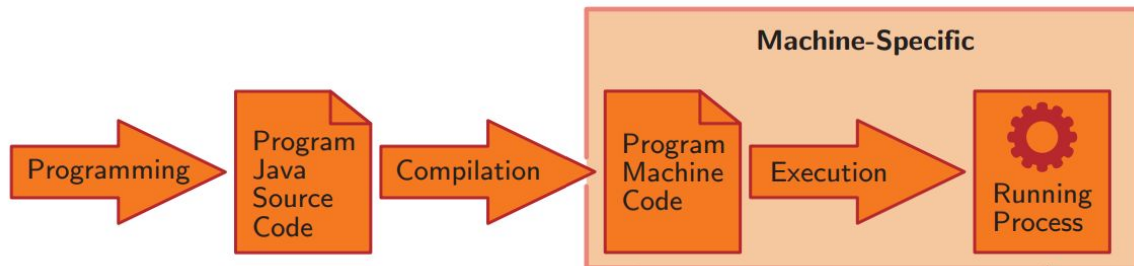
JIT компиляция

- JIT компилятор работает параллельно с интерпретацией
- Ищет компромисс между задержкой и скоростью выполнения
- В реализации HotSpot трехуровневая система исполнения байт-кода:
Интерпретатор + C1 + C2



АОТ компиляция

- Компиляция Java напрямую в машинный код
- Может себе позволить более “дорогие” оптимизации
- Минимизирует потребление ресурсов во время исполнения



JVM

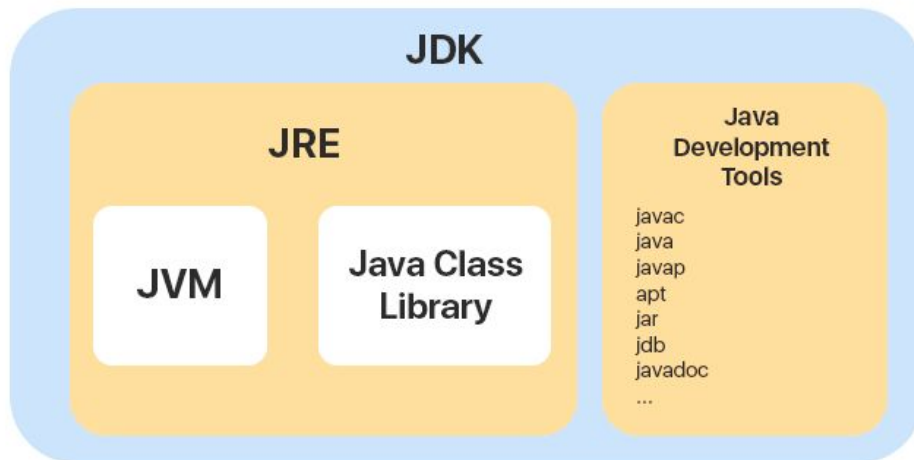
- JVM (Java Virtual Machine) имеет [спецификацию](#), согласно которой разрабатываются ее реализации под конкретную платформу.
- [HotSpot](#) – основная реализация JVM, выпускаемая компанией Oracle.

JVM, JRE, JDK

JVM (Java Virtual Machine) – виртуальная машина Java.

JRE (Java Runtime Environment) – среда исполнения Java.

JDK (Java Development Kit) – полнофункциональный SDK для Java.

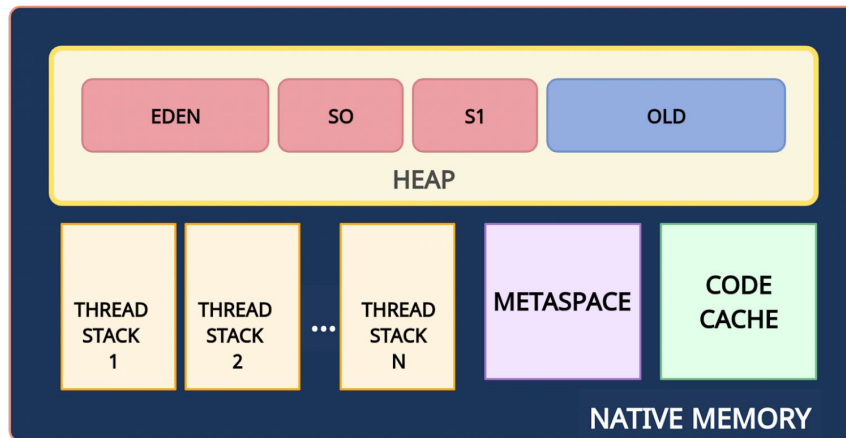


Принципы Java

- Простой понятный синтаксис C
- Поддержка ООП
- Кроссплатформенность
- Производительность
- Автоматическая сборка мусора
- Отсутствие низкоуровневого управления памятью
- Поддержка многопоточности

Сборка мусора

- Автоматический процесс
- Позволяет эмулировать бесконечную память
- Обеспечивает безопасность памяти
- В неопределенное для нас время останавливает приложение (Stop-The-World)



Hello World!

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```


Про системы сборки Java приложений

- Автоматическая сборка приложения
- Позволяет загружать сторонние зависимости из репозиториев артефактов
- Централизованное версионирование

Вопросы?

