Основы конструирования программ

Преподаватель:

к.т.н., доцент кафедры Инженерной психологии и эргономики
Меженная Марина Михайловна
mezhennaya@bsuir.by
а 606-2

Лекция 3: Методология структурного программирования

План лекции:

1.История возникновения и развития языков программирования.

2.Стили программирования: императивный и декларативный.

3.Парадигмы программирования: процедурная (структурная), объектно-ориентированная, функциональная, логическая.

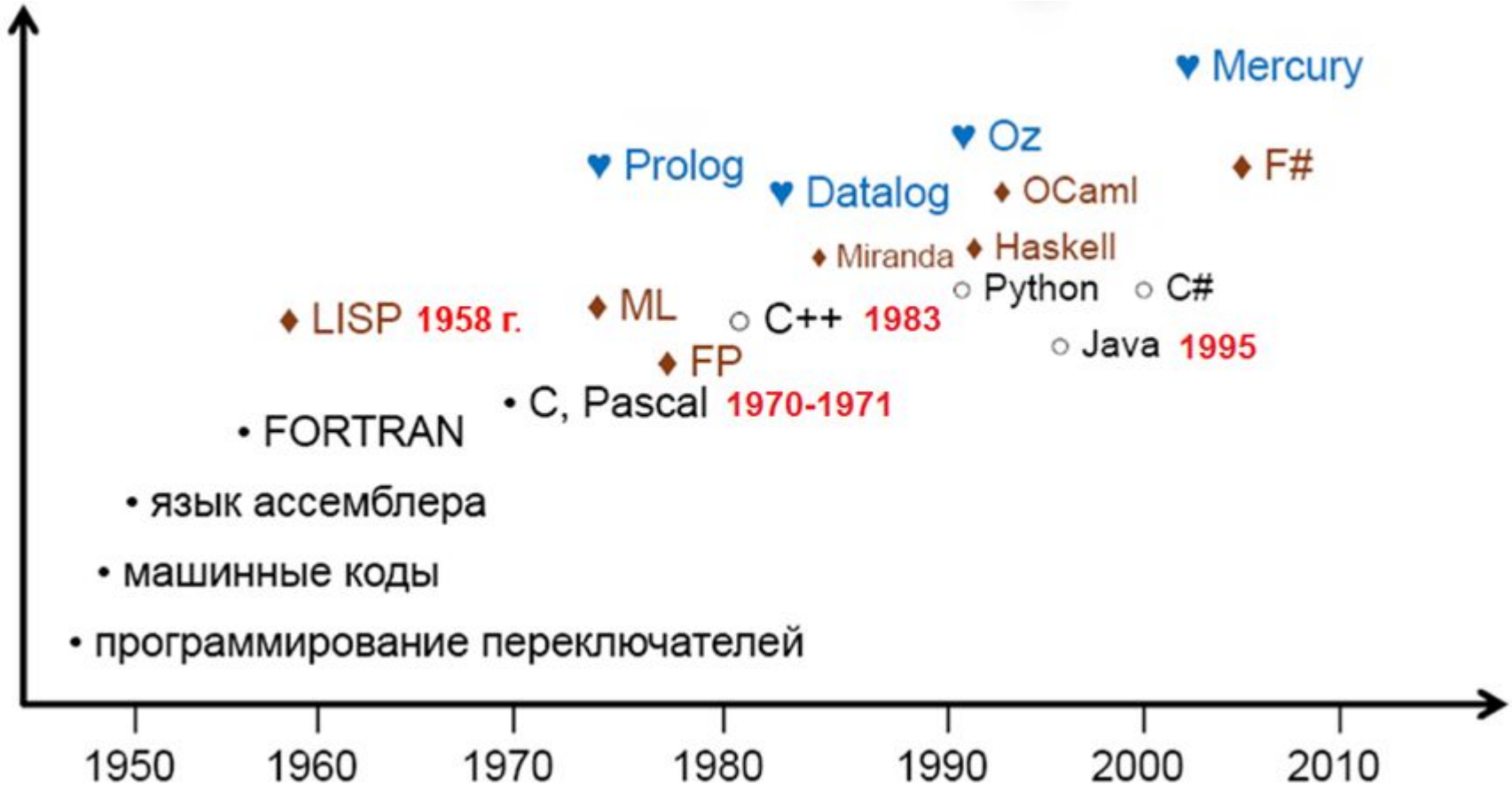
4.Объектно-ориентированное программирование.

5.Классификация языков программирования:
- по уровню абстракции,
- по структуре кода.

Языки программирования: исторический очерк

- Первый язык программирования высокого уровня – ФОРТРАН – был создан Дж.Бэкусом, чтобы математики могли программировать на уровне формул.

Языки программирования: исторический очерк



Стили программирования: императивный и декларативный.

Парадигмы программирования: процедурная (структурная), объектно-ориентированная, функциональная, логическая.

Стили программирования: императивный

Программируя в императивном стиле, программист должен объяснить компьютеру, **как нужно решать задачу.**

Императивная программа это последовательность команд / логических переходов, которые должен выполнить компьютер.

Основные конструкции:

- Манипулирование ячейками памяти
- Оператор присваивания
- Явные операторы передачи управления
- Условный оператор, циклы

Стили программирования: декларативный

Программа представляет собой совокупность утверждений, описывающих фрагмент предметной области или сложившуюся ситуацию; **описывается результат (его свойства), а не методы его достижения.**

Программист лишь описывает решаемую задачу, а поиском решения занимается императивная система программирования.
В итоге получаем значительно большую скорость разработки приложений, значительно меньший размер исходного кода.

Парадигмы программирования

Парадигма программирования - исходная концептуальная схема постановки проблем и их решения.

Вместе с языком, ее формализирующим, парадигма формирует **стиль программирования.**

Парадигма отражает то, как программист видит выполнение программы.

Например, в объектно-ориентированном программировании программист рассматривает программу как набор взаимодействующих объектов, тогда как в функциональном программировании программа представляется в виде цепочки вычисления функций.

Парадигмы программирования: процедурная

Программа состоит из структур данных (**объектов обработки**) и алгоритма (**метода обработки**).

Программист должен **в явном виде** описать все вычисления, которые должен проделать компьютер.

Для управления процессом выполнения используются следующие конструкции: последовательность, ветвление, цикл и вызов подпрограммы.

Эта парадигма является **самой старой.** Она развивалась по мере появления новых концепций в языках программирования: трансляция (Ассемблер, Fortran, Cobol), типизация (Pascal), модули (Modula), универсальность (C, Ada).

Парадигмы программирования: функциональная

Функциональные языки **оперируют данными.** Применение функции к аргументам изменяет данные.

Единственной управляющей конструкцией является **вызов функции.** При этом существует некоторое множество базовых функций, а все другие функции строятся из базовых функций с помощью композиции. Используется **рекурсия.**

Сформировалась как дань математической направленности при исследовании и развитии искусственного интеллекта.
В настоящее время существуют сотни функциональных языков программирования, ориентированных на разные классы задач и виды технических средств: Lisp, Haskell, …

Парадигмы программирования: логическая

Логическая программа оперирует **пространством поиска решений.**

Вместо алгоритма решения задачи описывается мир задачи: какие имеются объекты, их свойства и отношения между ними.

Программа представляет собой **набор отношений**, которые называются фактами, и **правил**, на основании которых могут быть получены новые отношения. Это своего рода база данных (БД). Ее применение инициализируется запросом. Поиск ответа на запрос заключается в попытке логического вывода запроса на основании фактов и правил, имеющихся в БД.

Возникло как упрощение функционального программирования для математиков и лингвистов, решающих задачи символьной обработки. Основной язык – Prolog.

Парадигмы программирования: объектно-ориентированная

В объектно-ориентированном подходе исходная задача представляется как **совокупность взаимодействующих объектов**.

По аналогии с реальным миром объект характеризуется совокупностью **свойств** (структур данных, характерных для этого объекта), **методов их обработки** (подпрограмм изменения свойств) и **событий**, на которые данный объект может реагировать и которые приводят, как правило, к изменению свойств объекта.

Объекты могут иметь идентичную структуру и отличаться только значениями свойств. В таких случаях в программе создается новый тип данных – **класс.** Каждый конкретный объект, имеющий структуру этого класса, называется **экземпляром класса**.

Базовые понятия ООП

Класс — это тип данных, вводимый пользователем.
Основное назначение класса - описание основных свойств и методов обработки объектов этого типа данных.
С точки зрения программирования класс можно рассматривать как набор данных (полей, атрибутов, членов класса) и функций для работы с ними (методов).

Объект (экземпляр класса) – это отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом.

То есть класс – это логическая конструкция, а объект обладает физической сущностью (т.е. объект занимает область в памяти).

Базовые понятия ООП

Базовые понятия ООП

Свойство - характеристика объекта, его параметр. Все объекты наделены определенными свойствами, которые в совокупности выделяют объект из множества других объектов.

Методы - функции, связанные с определенным объектом, осуществляющие преобразование свойств, изменяющие поведение объекта.

Событие - изменение состояния объекта. Внешние события генерируются пользователем (например, клавиатурный ввод или нажатие кнопки мыши, выбор пункта меню, запуск макроса). Внутренние события генерируются системой.

Базовые понятия ООП

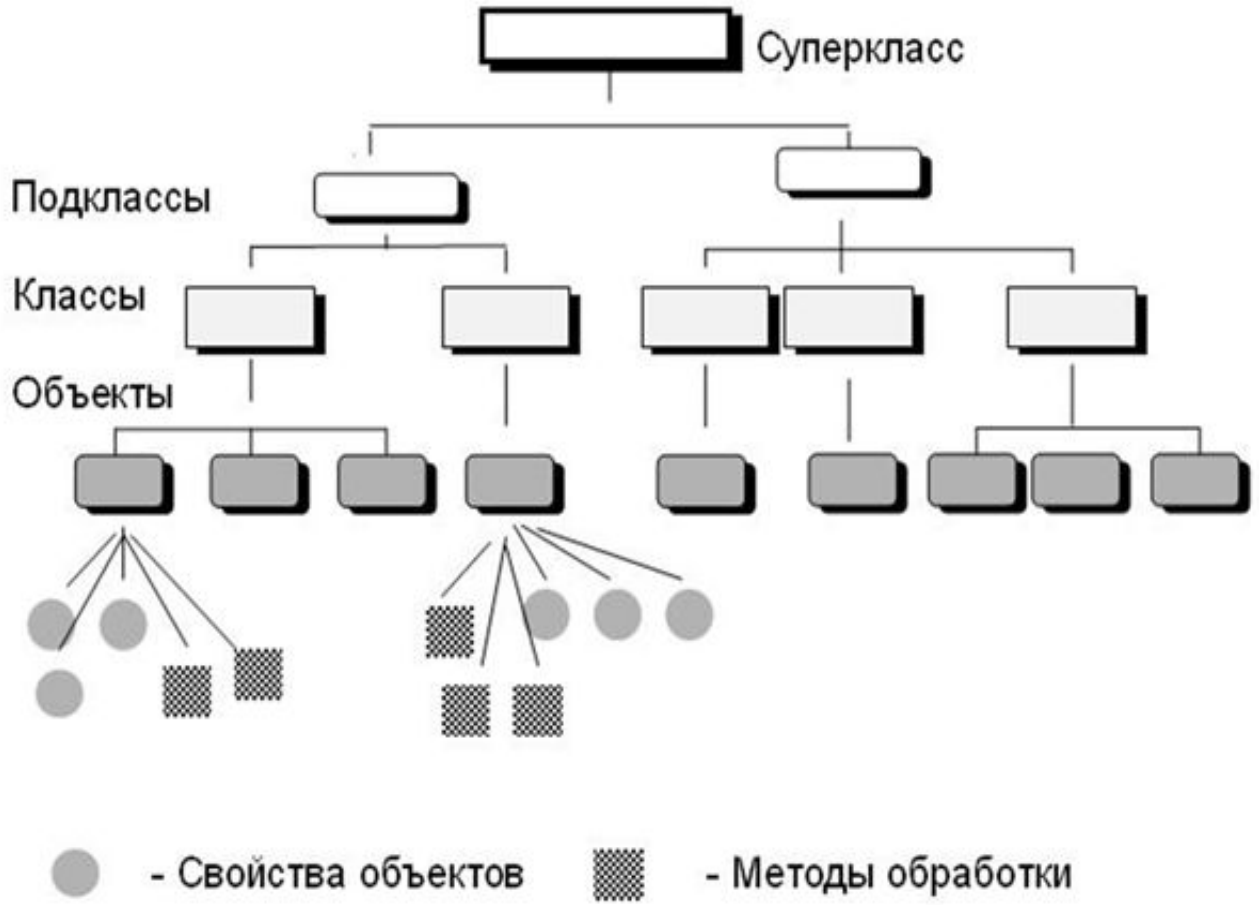
1.Абстракция данных – выделение только необходимых (существенных для поставленной задачи) характеристик и методов, которые в полной мере описывают объект.

2.Инкапсуляция – объединение данных и методов (функций) для работы с ними, а также сокрытие внутренней реализации класса от пользователя.

3.Наследование – механизм, позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса частично или полностью заимствуются новым классом.

4.Полиморфизм – возможность использования одних и тех же методов для работы с различными объектами базового и порожденного им классов.

Базовые принципы ООП



Достоинства ООП

- Абстракция от деталей реализации.
- Данные и операции описываются вместе.
- Модульность (локализация кода и данных улучшает наглядность).
- Защита наиболее критичных данных от несанкционированного доступа.
- Возможность создавать расширяемые системы.
- Возможность многоразового использования программных компонентов.

Недостатки ООП

- Сложность понимания концепций ООП.
- Сложность проектирования и использования классов.
- Излишняя универсальность.
- Неэффективность на этапе выполнения (инкапсуляция данных приводит к необходимости выполнения процедурного вызова каждый раз при доступе к данным).

Процесс объектно-ориентированного проектирования

Этап анализа задачи:

1.Осуществляется идентификация объектов и их свойств;

2.Устанавливается перечень операций (методов обработки), выполняемых над каждым объектом, в зависимости от его состояния (событий);

3.Определяются связи между объектами для образования классов.

4. Устанавливаются требования к интерфейсу с объектами.

Этап детального проектирования:

1.Разрабатывается структура классов, описывающая связь между классами и объектами;

2.Разрабатываются диаграммы объектов, показывающие взаимосвязи с другими объектами;

3.Разрабатывается внутренняя структура программного продукта.

Процедурное программирование или ООП?

1.Является ли **проблема** статичной или динамичной (буду ли изменения и новые версии)? ООП для статической проблемы будет мешать, т.к. не показывает всю картину целиком.

2.Является ли Ваше **решение** одно- или многоуровневым? ООП хорош, если необходимо разбивать проблему сверху вниз и нужно задумываться о каждом конкретном шаге.

3.Глобальное или локальное **состояние** программы? Состояние определяется статусом переменных. Если везде статические переменные, то зачем нужен ООП?

Классификация языков программирования по уровню абстракции

1 уровень: машинный

2 уровень: машинно-ориентированный

машинно-зависимые (низкого уровня)

3 уровень: структурные (процедурно-ориентированные и объектно-ориентированные)

4 уровень: проблемно-ориентированные

5 уровень: ???

машинно-независимые (высокого уровня)

Классификация языков программирования по структуре кода

1.Неструктурные (используют операторы goto или jump: – языки машинного и машинно-ориентированного уровней)

2.Структурные

3.Объектно-ориентированные

4.Визуальные