PARADIGMAS DE PROGRAMAÇÃO

01 - Paradigma Imperativo

(Década de 1950)

- Nesse paradigma, os programas são construídos em torno de instruções que mudam o estado do programa. A ênfase está nas ações a serem executadas e nas sequências de passos.
- Exemplo: Assembly, Fortran, C.
- Notas: Foi o primeiro paradigma a surgir, focando nas operações básicas do hardware.

PARADIGMAS DE PROGRAMAÇÃO

02 - Paradigma Estruturado

(Década de 1960)

- Baseado na ideia de modelar o mundo real através de "objetos" que encapsulam dados e funções relacionadas. Promove reutilização e modularidade.
- Exemplos: Simula, Smalltalk, Java, C++.

- **Vantagens do Paradigma Estruturado:**

- Código organizado em estruturas de controle.
- Fácil legibilidade e compreensão.
- Manutenção simplificada.
- Controle direto sobre o fluxo de execução.

- **Desvantagens do Paradigma Estruturado:**

- Reutilização limitada de código.
- Dificuldade em lidar com problemas complexos.
- Menos escalável para projetos grandes.
- Menos intuitivo para certos domínios específicos.

PARADIGMAS DE PROGRAMAÇÃO

03 - Paradigma Funcional

(Década de 1950/1960)

- O foco está na avaliação de funções matemáticas. Evita estados mutáveis e enfatiza a imutabilidade dos dados.
- Exemplos: Lisp, Haskell, Erlang.
- Notas: Inspirado em cálculos lambda e lógica matemática, promove abstração poderosa.

PARADIGMAS DE PROGRAMAÇÃO

04 - Paradigma Orientado a Objetos

(Década de 1960/1970)

- O programa é construído a partir de componentes independentes e reutilizáveis. Esses componentes podem ser conectados para criar sistemas mais complexos.
- Exemplos: COM/DCOM, CORBA.
- Notas: Busca promover a reutilização e interconexão flexível de partes do software.

- **Vantagens do Paradigma Orientado a Objetos:**

- Reutilização de código através de herança e composição.
- Modularidade que facilita a manutenção e evolução.
- Modelagem natural de entidades do mundo real.
- Polimorfismo para tratar objetos de diferentes classes.
- Proteção dos dados através da encapsulação.

- **Desvantagens do Paradigma Orientado a Objetos:**

- Complexidade com hierarquias de classes excessivamente profundas.
- Possível overhead de memória em comparação com abordagens mais simples.
- Aprendizado desafiador para iniciantes.
- Performance mais lenta em certos casos.

PARADIGMAS DE PROGRAMAÇÃO

05 - Paradigma Lógico

(Década de 1970)

- Os programas consistem em fatos e regras lógicas que são usados para inferir resultados. Lida com problemas de busca e dedução.
- Exemplos: Prolog.
- Notas: Baseado na lógica formal e modelagem de conhecimento declarativo.

PARADIGMAS DE PROGRAMAÇÃO

06 - Paradigma Imperativo

(Década de 1980)

- O programa define regras que são aplicadas a dados para realizar ações. Focado na tomada de decisão baseada em condições.
- Exemplos: OPS5, CLIPS.
- Notas: Usado em sistemas de produção e sistemas especialistas.

PARADIGMAS DE PROGRAMAÇÃO

07 - Paradigma Orientado a Aspectos

(Década de 1990)

- Separação das preocupações (aspects) do sistema em módulos independentes. Permite modularizar características transversais.
- Exemplos: AspectJ.
- Notas: Combate a duplicação de código e melhora a modularidade.

PARADIGMAS DE PROGRAMAÇÃO

08 - Paradigma Orientado a Componentes

(Década de 1990)

- O programa é construído a partir de componentes independentes e reutilizáveis. Esses componentes podem ser conectados para criar sistemas mais complexos.
- Exemplos: COM/DCOM, CORBA.
- Notas: Busca promover a reutilização e interconexão flexível de partes do software.

PARADIGMAS DE PROGRAMAÇÃO

09 - Paradigma Reativo

(Década de 1990)

- Lida com fluxos de eventos e reage a mudanças assíncronas. Ênfase na propagação de mudanças e na escalabilidade.
- Exemplos: ReactiveX, Akka.
- Notas: É fundamental para sistemas que lidam com alta concorrência e interações assíncronas.

PARADIGMAS DE PROGRAMAÇÃO

10 - Paradigma Orientado a Serviços

(Década de 2000)

- Componentes independentes oferecem serviços através de interfaces padronizadas, geralmente em redes distribuídas.
- Exemplos: Web services, SOA.
- Notas: Promove interoperabilidade entre sistemas heterogêneos.