Aluno: Christian Amarildo Amorim Morais

Disciplina: Paradigmas de linguagem de programação

1) Facilita o desenvolvimento ao permitir dividir o problema em subpartes, deixando o código mais compreensível, reutilizável e fácil de manter. Permite padronização, planejamento preciso, escalabilidade e localização de erros de forma mais eficiente.

2) Escopo é onde uma variável pode ser acessada. Tempo de vida é o período em que ela existe na memória. a) Variáveis globais: escopo global alocadas em memória estática. b) Variáveis locais: escopo local, alocadas na pilha. c) Parâmetros formais: também na pilha, com escopo local à função. d) Heap: alocados dinamicamente via malloc/new, controle manual do tempo de vida e acesso via ponteiros.

3) Por valor: cópia do dado é passada; alterações não afetam o original. Por referência: o endereço é passado; alterações afetam o original.

Exemplo por ~~valor~~ parâmetro:

```
void dobrar(int x) { x *= 2; }
```

Exemplo por referência

```
void dobrar(int *x) { *x *= 2; }
```

4) É uma combinação de valores, variáveis e operadores que retornam um resultado.

Tipos:

Aritmética: 3 + 5 Lógica: a && b

Relacional: x > y Função: soma(2,3)

~~5) Legibilidade, facilidade de escrita, confiabilidade (verificação de tipos, tratamento de erros) e baixo custo de desenvolvimento e manutenção.~~

Aluno: Christian Amarildo Amorim morais
Disciplina: Paradigmas de linguagem de programação

1) Facilita o desenvolvimento ao permitir dividir o problema em subpartes, deixando o código mais compreensível, reutilizável e fácil de manter. Permite padronização, planejamento preciso, escalabilidade e localização de erros de forma mais eficiente.

2) Escopo é onde uma variável pode ser acessada. Tempo de vida é o período em que ela existe na memória. a) Variáveis globais: escopo global alocadas em memória estática. b) Variáveis locais: escopo local, alocadas na pilha. c) parâmetros formais: também na pilha, com escopo local à função. d) Heap. alocados dinamicamente via malloc/new, controle manual do tempo de vida e acesso via ponteiros.

3) Por valor: cópia do dado é passada; alterações não afetam o original. Por referência: o endereço é passado; alterações afetam o original.

Exemplo por ~~valor~~ parâmetro:

```
void dobrar( int x) { x *= 2 ; }
```

Exemplo por referência

```
void dobrar (int *x) { *x *=2; }
```

4) É uma combinação de valores, variáveis e operadores que retornam um resultado.

Tipos:

Aritmética: 3+5 | Lógica: a && b
Relacional: x > y | Função: soma(2,3)

~~5) Legibilidade, facilidade de escrita, confiabilidade (verificação de tipos, tratamento de erros) e baixo custo de desenvolvimento e manutenção.~~

6) Tipos controlam quais valores são permitidos, evitando erros. Em linguagens como Python, apesar da flexibilidade, perde-se robustez por não haver verificação de tipos em tempo de compilação.

7) Arrays são semelhantes a funções totais de índices para valores: A[i] = valor. Ex. A(0)=1

Registros são produtos cartesianos etiquetados (tuplas nomeadas). Array se assemelham a funções.

8) É o total de valores que um tipo pode representar. Array de 3 inteiros com 100 valores → 100^3 = 1.000.000. Registro com nome (256^{50}), idade (100), nota (100) → produto das cardinalidades.

9) Tipos definidos em termos de si mesmos, usados para listas e árvores.

Ex:

```
struct Nodo {
    int chave;
    struct Nodo *esq, *dir;
};
```

10)
1. Deve haver um caso base.
2. Passo recursivo deve ocorrer.
3. Cada chamada deve se aproximar do caso base.
4. Cada chamada é empilhada na stack.

11) Simplicidade e clareza na solução de problemas complexos com chamadas auto-referenciais.

12) ~~Com~~ Quando há muitos dados ou chamadas profundas, pode causar estouro de pilha (stack overflow) e perda de desempenho.

13) Modelos conceituais com operações definidas e escondem a implementação interna. Ex: pilha (stack) com ~~operadores~~ operações push, pop, top, isEmpty.

14)

. Abstração: focar no essencial.

. Encapsulamento: esconder detalhes internos.

. Herança: reuso e especialização

. Polimorfismo: mesma operação com comportamentos diferentes.

E os benefícios: modularidade, reuso, clareza e facilidade de manutenção.

15) Evita acesso indevido a dados. Métodos controlam o acesso (getters/setters). Flexibilidade é possível: atributos simples podem ser públicos para evitar código excessivo.

16)

~~- Dependência: uma classe usa outra. Ex: método recebe outr objeto~~

~~- Extensão: herança~~

A dependência ocorre quando uma classe utiliza outra em alguma parte de seu código, geralmente como parâmetro ou variável local de um método. Nesse caso, a classe depende da outra para realizar alguma tarefa.

Ex:
```
class Impressora {
        void imprimir (Documento doc) {
            System.out.println("Imprimindo: " + doc.getTexto());
            }
        }
```

A extensão é o relacionamento de herança entre classes, no qual uma classe (subclasse) herda atributos e métodos de outra (superclasse) como no caso das especializações ou heranças.

Ex.
```
  class Animal {
      void fazerSom() {
          System.out.println("Som genérico");
          }
      }
  class Cachorro extends Animal {
      void fazerSom() {
          System.out.println("Au Au");
      }
  }
```

A inclusão representa a composição ou agregação de objetos, ou seja, uma classe possui instâncias de outras classes como parte de sua estrutura interna.

Ex:

```
class motor{
    void ligar(){
        system.out.println("motor ligado");
    }
}

class carro{
    private motor motor = new motor();

    void ligar Carro(){
        motor.ligar();
    }
}
```

17) Acoplamento é ~~defini~~ definido como o grau de interdependência entre duas classes ou módulos. O acoplamento forte ocorre quando uma classe conhece muitos detalhes da outra, dependendo fortemente de sua estrutura interna ou de mudanças específicas em sua implementação. O acoplamento fraco acontece quando as classes interagem entre si por meio de interfaces bem definidas, conhecendo apenas o necessário umas das outras.

Quando se cria uma hierarquia de classes, uma herança mal planejada pode levar a um acoplamento forte, especialmente quando subclasses dependem profundamente do comportamento ou da estrutura interna da superclasse, fazendo com que mudanças na superclasse resultem na propagação de efeitos problemas inesperados nas subclasses, prejudicando a estabilidade e dificultando a extensão do sistema

18) Em POO o foco está na modelagem de objetos que representam entidades com estado e comportamento, encapsulando dados e operações. O sistema é estruturado como uma colaboração entre objetos que interagem por meio de métodos. Já na programação imperativa, o foco está em comandos sequenciais que instruem o computador passo a passo, com dados e funções geralmente separados. Essa separação pode aumentar a complexidade e dificultar a manutenção, especialmente em sistemas grandes, onde o acoplamento tende a crescer e a reutilização de código é mais limitada. Por outro lado, a POO, com sua modularidade e encapsulamento, facilita a divisão do sistema em partes menores e independentes o que favorece a ~~compreensão~~ compreensão, manutenção e reutilização. Assim, a arquitetura

orientado a objetos é mais adequado para grandes sistemas, por oferecer melhor organização, escalabilidade e manutenção a longo prazo.

19) Um dos principais desafios ~~foi~~ foi manter a compatibilidade com C, permitindo que programas escritos em C pudessem ser compilados como C++ sem exigir muitas modificações. Isso exigiu cuidado extremo para que os novos recursos de C++ não interferissem com o comportamento do C, o que impôs restrições de sintaxe e semântica na linguagem C++. Outro desafio importante foi integrar os fundamentos da orientação a objetos, como abstração, encapsulamento, herança e polimorfismo, sem sacrificar ~~o [illegible] de~~ o desempenho e o controle fino sobre recursos que o C proporciona. Como C é uma linguagem de baixo nível, próxima ao hardware, o C++ precisou oferecer suporte a objetos sem esconder completamente o funcionamento interno, o que ~~[illegible]~~ levou à criação de mecanismos ~~[illegible]~~ como construtores, destrutores, sobrecarga de operadores e ponteiros para funções virtuais, todos implementados com atenção ao custo de execução.

20) Semântica determina o que ocorre quando um objeto atribuído a outro ou como parâmetro. A semântica de cópia cria um objeto novo e independente, alterações nele não afetam o original, já na referência cria apenas um apelido que aponta para o objeto, ~~[illegible]~~ modificações afetam o original, no C++ isso é feito com referência direta, com &

5)
Legibilidade: a LP deve possuir elementos de fácil entendimento e não-ambíguos contendo número reduzido de elementos básicos, instruções de controles claras, sintaxe limpa e facilidade para representação de estruturas de dados

Facilidade de escrita: ser simples e ter suporte a abstrações permitindo a representação resumida de objetos

Confiabilidade: as implementações dos problemas devem reproduzir os resultados esperados, utilizando mecanismos de verificação de tipos e manipulação de ~~dados~~ exceções

Custo: é o custo de ~~[illegible]~~ desenvolvimento, treinamento, aprendizado, manutenção, ferramentas e suporte, que são necessários para manter o software. Linguagens legíveis têm custos menores