Introdução à Programação

Programação Imperativa
(Registros X Classes e Métodos Nativos)

Tópicos da Aula

- Hoje aprenderemos como podemos modelar algo parecido com classes em linguagens imperativas
  - Tipos estruturados
  - Manipulando estruturas
  - Referências para estruturas
  - Alocando memória dinamicamente
  - Garbage Collection
  - Usando Unions para simular Herança
- Depois como podemos integrar programas em Java a funções em C
  - Métodos Nativos
  - Exemplo de utilização de métodos nativos

Ao invés de classes, estruturas (registros)!

```
struct Conta{
   double saldo;
   char numero[11];
};
```

Define um tipo, sem information hiding, sem funções, sem Construtor, sem herança, sem subtipo

Lista-se apenas os componentes de cada elemento do tipo

Criando estruturas estaticamente

Cria e inicializa

Cria mas não inicializa

Acesso e Atualização de Partes da estrutura

```
struct conta c = {0,"1"};

struct conta d;

d.saldo = c.saldo;
strcpy(d.numero,"2");
```

Função da biblioteca padrão de C

Manipulando estruturas

```
struct Conta creditar(struct conta x,double v){
   x.saldo = x.saldo + v;
   return x;
}
```

```
struct conta c = {0,"1"};
struct conta d;
d = c;
c = creditar(c,10);
```

A estrutura é copiada, não trabalha-se com referências

A passagem de parâmetros é por valor: é necessário retornar o resultado

Manipulando referências para estruturas

```
void creditar(struct Conta *c,double v){
  (*c).saldo = (*c).saldo + v;
}
```

this

A estrutura é copiada

```
struct conta c = {0,"1"};
struct conta d;
d = c;
creditar(&c,10);
```

Passa-se a referência; os efeitos da execução do método são refletidos em c

Notação especial para manipular referências para estruturas

```
void creditar(struct Conta *c,double v){
   c -> saldo = c -> saldo + v;
}
```

Estruturas complexas

```
struct Endereco{
   char rua[40];
   char complemento[10];
   char cep[10];
   char cidade[20];
   char estado[20];
};

struct Pessoa{
   char nome[35];
   struct Endereco *endereco;
   struct Pessoa *conjuge;
};
```

Permitindo compartilhamento de estruturas e recursão

Criando estruturas dinamicamente

```
struct Conta *pc;

pc = (struct Conta *)
     malloc(sizeof(struct Conta));
```

Não cria a estrutura mas sim uma variável ponteiro

Cast

Cria a estrutura: aloca memória dinamicamente

Indica o espaço necessário para armazenar um elemento do tipo

Simulando **new**

```
struct Conta *newConta(char *num,double v) {
  struct Conta *retorno;
  retorno = (struct Conta *)
            malloc (sizeof(struct Comta));
  if (retorno == NULL){
    fprintf("Erro na alocação de memória!")
  } else {
    strcpy(retorno->numero, num);
    retorno->saldo = v;
  }
  return retorno;
}
```

Gerando lixo

```
struct Conta *pc, *pd;
pc = (struct Conta *)
     malloc(sizeof(struct Conta));
pd = (struct Conta *)
     malloc(sizeof(struct Conta));
pc = pd;
```

A primeira estrutura criada não pode ser mais acessada, vira lixo! A memória não será liberada...

Eliminando lixo

```
struct Conta *pc, *pd;
pc = (struct Conta *)
     malloc(sizeof(struct Conta));
pd = (struct Conta *)
     malloc(sizeof(struct Conta));
free(pc);
pc = pd;
```

O programador é responsável pelo gerenciamento da memória da primeira estrutura criada

Ao invés de supertipo, unions

```
union ContaPoupanca {
 struct Conta c;
 struct Poupanca p;
}

union ContaPoupanca cp;
cp.c = conta;
cp.p = poupanca;
```

Os elementos desse tipo podem ser tanto conta quanto poupança

Perde-se a conta armazenada em cp

Simulando instanceof

```
struct ContaGeral{
 int tipoContaGeral;
 union ContaPoupanca contaGeral;
}

struct ContaGEral cg;...
If (cg.tipoContaGeral == 0){
  cg.contaGeral.c.saldo = 0;
} else {
  cg.contaGeral.p.juros = 0;
}
```

Indica o tipo da informação armazenada na union contaGeral

Uma conta Armazenada?

Métodos Nativos

- Não é raro termos que integrar sistemas diferentes
  - Sistemas podem estar implementados em linguagens diferentes
- Java oferece um conjunto de ferramentas para integrar programas escritos em Java a outros escritos em C, C++ e Assembly
  - Java Native Interface (JNI)
  - Importante para reutilização de sistemas legados
  - Útil para chamar programas que interagem diretamente com periféricos

Métodos Nativos

- Um método que chama uma função implementada em outra linguagem é chamado de método nativo
  - O método é declarado com a palavra reservada **`native`**
- Só o cabeçalho do método é declarado
  - Implementação é feita pela função escrita em outra linguagem de programação

Exemplo: Classe HelloWorld

```
public class HelloWorld {

    public native void sayHello();

    static {
        System.load("hello.lib");
    }
    public static void main(String[] args) {

        HelloWorld h = new HelloWorld();

        h.sayHello();

    }
}
```

Inicialização estática para carregar biblioteca dinâmica de C

HelloWorld: Criando Header para Arquivo C

```
#include <jni.h>
/*
 * Class:     HelloWorld
 * Method:    sayHello
 * Signature: ()V
 */
JNIEXPORT void JNICALL Java_HelloWorld_sayHello
  (JNIEnv * env, jobject obj);
```

Arquivo Header de C pode ser gerado Automaticamente através de javah

HelloWorld: Escrevendo a implementação

```
#include <stdio.h>
#include "HelloWorld.h"

JNIEXPORT void JNICALL Java_HelloWorld_sayHello
  (JNIEnv * env, jobject obj) {

    printf("\nHelloWorld!!!\n");

}
```

Resumindo

- Tipos Estruturados (Registros)
  - Manipulando estruturas
  - Referências para estruturas
  - Alocando memória dinamicamente
  - Garbage Collection
  - Usando Unions para simular Herança
- Métodos Nativo
  - Exemplo de utilização de métodos nativos