Variáveis Globais

- As variáveis globais precisam ser visíveis em todo o programa, e seu alcance é toda a execução do mesmo
- Não faz sentido armazená-las em um registro de ativação
- Elas possuem um endereço fixo no espaço de memória do programa
- O endereço real da global na memória vai ser determinado no momento da carga do programa, pelo *loader* do sistema operacional

Alocação Dinâmica

- Existem valores cujo alcance pode ser maior do que o das variáveis que possuem *ponteiros* para eles:

```
static int* foo() {
  int *foos = (int*)malloc(10 * sizeof(int));
  return foos;
}
```

```
Foo foo() {
  return new Foo();
}
```

- O vetor e o objeto alocados dentro da função e do método *foo* precisam sobreviver ao registro de ativação da chamada a *foo*

- Esses valores não são armazenados na pilha, mas ficam em outra área da memória chamada *heap*

- A recuperação da memória no heap depois que o alcance dos valores termina pode ser *manual* (como em C, usando *free*), ou *automática* (como em Java, usando um coletor de lixo ou contagem de referências)

Alinhamento

- A memória de um computador moderno pode ser dividida em blocos de 4 ou 8 bytes, a depender do tamanho da *palavra* do processador (32 ou 64 bits), mas os endereços de memória são contados em *bytes*

- Muitas máquinas ou não podem acessar endereços que não são *alinhados* com o início desses blocos, ou pagam um preço em desempenho nesses acessos

- É responsabilidade do compilador evitar acessos não-alinhados, em geral garantindo que os endereços das variáveis respeitam o alinhamento

- Algumas plataformas podem ter regras de alinhamento mais exóticas: em x64 o local no AR onde o endereço de retorno é armazenado tem que ser alinhado a blocos de *16* bytes

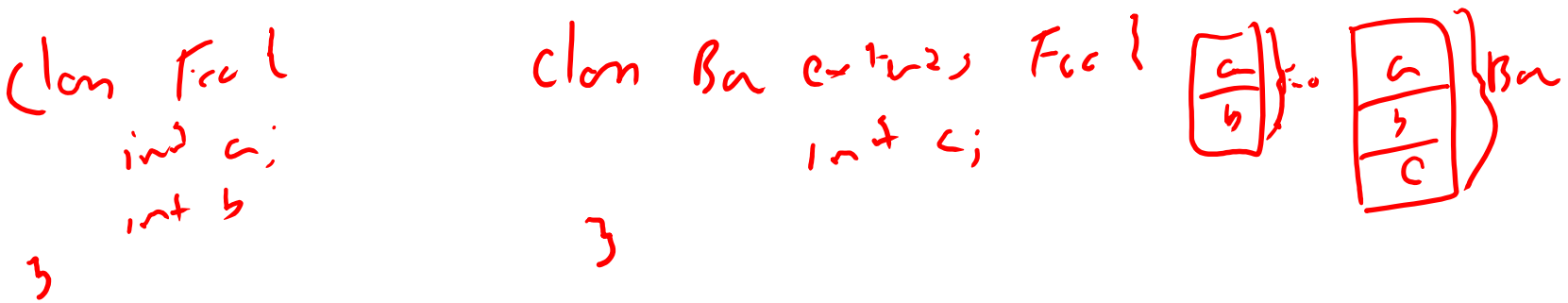
Objetos e herança

- Geralmente, os campos de um objeto são dispostos sequencialmente na memória

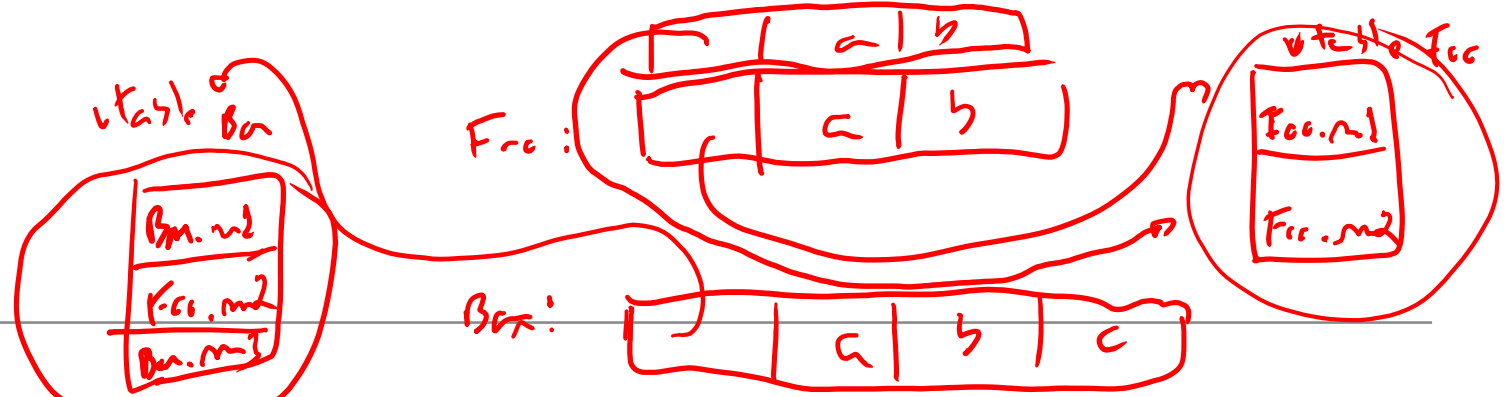
- Se *f* é uma instância da classe *Foo*, e *b* uma instância da classe *Bar* que extende *Foo*, então os os campos que *b* tem em comum com *f* aparecem nas mesmas posições, e só depois vêm os campos específicos de *Bar*

- Isso permite que qualquer instância de *Bar* funcione como uma instância de *Foo* quando acessamos seus campos

- Mas métodos são diferentes, por causa da redefinição: precisamos de *vtables*



Vtables



- Uma vtable é um vetor que contém o endereço de todos os métodos de uma classe
- A vtable de uma subclasse começa igual à de sua superclasse direta
  - Novos métodos aumentam a vtable
  - Métodos redefinidos substituem entradas que já estão na vtable
- Todo objeto contém um ponteiro para a vtable de sua classe
- Toda chamada a método usa a vtable do objeto