01 Ciclo de Vida de um Objeto

Transcrição

Neste passo, abordaremos o ciclo de vida de um objeto:

- O momento de sua criação;
- Quando não é referenciado;
- E quando não está mais na memória, ou foi *garbage collected*.

Para decifrarmos essas três situações, entenderemos algumas regras bem definidas.

O primeiro passo é vermos quando um objeto é criado; Já temos um arquivo com `Carro`, e criaremos outro para uma nova classe de teste chamada `TestaCicloDeVida` que terá um método `main()`.

Dentro deste, definiremos uma variável `c` do tipo `Carro`. Se rodarmos este código, teremos a pergunta de certificação: "Quantos carros foram criados?".

```
class TestaCicloDeVida {
    public static void main(String[] args) {
        Carro c;
    }
}
```

COPIAR CÓDIGO

No caso desta questão simples, a resposta é zero pois não criamos nenhum carro. Para fazermos isso, precisaríamos chamar o construtor.

Para a certificação, o importante é sabermos que devemos chamar o construtor para criar o objeto.

Existem situações muito específicas com técnicas complexas em que conseguimos construir o objeto sem chamar o construtor, mas isso não será cobrado na certificação, pois não é um tópico tradicional.

Nossa real questão diz respeito à criação de um objeto tanto para certificação quanto para o dia-a-dia; para isso, sempre precisaremos usar `new`.

```
class TestaCicloDeVida {
    public static void main(String[] args) {
        Carro c = new Carro();
```

```
    }
}
```

COPIAR CÓDIGO

Respondendo à mesma pergunta anterior, a resposta seria que estamos criando apenas um carro; o `Carro()` foi criado e uma referência para este objeto foi colocada na variável `c`.

Portanto, a variável `c` não é um `Carro()`, e sim apenas referencia o objeto na memória que por sua vez é um `Carro()`.

Eventualmente falamos em "ponteiro" ou que "`c` é um `Carro()`" ao invés de "referenciar", porém é uma maneira informal de dizer que `Carro()` é um objeto que está na memória e que não podemos pegar, sendo referenciado através da variável `c`.

Se lembrarmos disso, várias das questões de referência de objetos serão resolvidas tranquilamente.

Adicionando mais uma linha somente com `new Carro();`, a resposta adequada à pergunta seria de que temos dois carros criados no código, um na linha 3** e outro na **4.

```
class TestaCicloDeVida {
    public static void main(String[] args) {
        Carro c = new Carro();
        new Carro();
    }
}
```

De volta ao código em que criamos apenas um `Carro()`, poderemos referenciá-lo e adicionar um valor qualquer como `2014` para `c.ano` e `"Ferrari"` para `c.modelo` por exemplo. Afinal, enquanto acessarmos uma variável que possui referência para o objeto, este se tornará acessível.

COPIAR CÓDIGO

Se nenhuma variável aponta mais para o objeto, não teremos como acessá-lo.

Para entendermos melhor, faremos um exemplo escrevendo somente `c` sendo igual a `new Carro()` para termos um outro carro, ou seja, outro objeto na memória.

```
class TestaCicloDeVida {
    public static void main(String[] args) {
        Carro c = new Carro();
        c.ano = 2014;
        c.modelo = "Ferrari";

        c = new Carro();
    }
}
```

COPIAR CÓDIGO

Em nossa memória, tínhamos `c` do tipo `Carro` igual a `new Carro()` que criou o primeiro carro, cujo ano é 2014 e o modelo é Ferrari.

Já na linha seguinte com somente `c` igual a `new Carro()`, criamos um novo carro na memória e mandamos `c` referenciar este novo objeto. Com isso, nenhuma variável estará mais

fazendo referência ao carro anterior, tornando-o inacessível.

Portanto, conseguiremos acessar somente o segundo carro até este ponto, pois o primeiro não está mais acessível pela `thread` do nosso programa.

Com isso, logicamente este objeto não tem mais serventia alguma e pode ser excluído, visto que não podemos acessá-lo.

A certificação pode questionar quantos objetos são acessíveis até certo ponto, e deveremos saber identificar essas quantidades corretamente.

Inserindo `c` igual a `null`, pegaremos a nossa referência que já tinha apontado para dois carros diferentes e faremos com que a variável não referencie mais nada.

```
class TestaCicloDeVida {
    public static void main(String[] args) {
        Carro c = new Carro();
        c.ano = 2014;
        c.modelo = "Ferrari";

        c = new Carro();

        c = null;
    }
}
```

COPIAR CÓDIGO

Desta forma, teremos dois objetos inacessíveis do tipo `Carro`.

Outra situação bem comum em que podemos ter objetos inacessíveis é quando criamos as variáveis e os objetos que serão referenciados por estas dentro de um bloco menor.

Aqui, colocamos dentro de um método. Se tivermos um `if()` com `15` sendo maior do que `10`, saberemos que o resultado será sempre verdadeiro. Dentro, escreveremos que `c2` do tipo `Carro` é igual a um novo carro, cujo `c2.ano` será `2010`.

```
class TestaCicloDeVida {
    public static void main(String[] args) {
        Carro c = new Carro();
        c.ano = 2014;
        c.modelo = "Ferrari";

        c = new Carro();

        c = null;

        if(15 > 10) {
            Carro c2 = new Carro();
            c2.ano = 2010;
        }
    }
}
```

COPIAR CÓDIGO

Dentro deste laço, a `c2` é acessível. Lembrando que o escopo de uma variável local é o bloco em que foi definida, esta se tornará inacessível fora de `if()`.

A partir do instante em que `if()` termina, o objeto que tinha

sido referenciado pela variável `c2` também não estará mais acessível.

Com isso, os três carros que foram criados estão inacessíveis.

É importante lembrarmos que esta regra vale para o laço `for()`; se definirmos uma variável dentro e criarmos um objeto, pode ser que não possa ser acessada de fora também.

Para termos um exemplo, criaremos dez carros; dentro de `for()`, adicionaremos `(int i=0;i < 10; i++)` para que, enquanto `i` for menor do que `10`, falaremos que a variável `c3` do tipo `Carro` é igual a um novo `Carro()`.

```
class TestaCicloDeVida {
    public static void main(String[] args) {
        Carro c = new Carro();
        c.ano = 2014;
        c.modelo = "Ferrari";

        c = new Carro();

        c = null;

        if(15 > 10) {
            Carro c2 = new Carro();
            c2.ano = 2010;
        }

        for(int t=0;i < 10; i++) {
            Carro c3 = new Carro();
        }
```

```
    }
}
```

COPIAR CÓDIGO

Até este ponto, criamos dez carros que estão inacessíveis fora de `for()`, visto que todos foram gerados somente dentro deste.

Porém, pode ser que a certificação pergunte a quantidade de objetos acessíveis se houver uma variável `c3` antes e fora de `for()`.

```
class TestaCicloDeVida {
    public static void main(String[] args) {
        Carro c = new Carro();
        c.ano = 2014;
        c.modelo = "Ferrari";

        c = new Carro();

        c = null;

        if(15 > 10) {
            Carro c2 = new Carro();
            c2.ano = 2010;
        }

        Carro c3;
        for(int t=0;i < 10; i++) {
            c3 = new Carro();
        }
    }
}
```

Sabemos que os três primeiros carros continuam inacessíveis. Porém, a variável `c3` ainda existe, já que foi declarada fora do *loop* `for()`. Também está referenciando o último carro criado, então existe apenas um carro acessível, enquanto os dez que foram gerados no laço permanecem sem acesso fora de `for()`.

COPIAR CÓDIGO

Portanto, devemos sempre chamar o construtor para criar um objeto com `new`.

Sabemos que um objeto é acessível enquanto for referenciado direta ou indiretamente, e se torna inacessível quando não tivermos mais referências diretas ou indiretas ao objeto, e podemos jogá-lo fora a qualquer instante com o Garbage Collector.

Por exemplo, quando criamos o segundo carro e o atribuímos para a variável `c`, o primeiro se tornou inacessível; é a partir disso que o objeto poderá ser destruído a qualquer momento.

Não quer dizer que necessariamente isso irá acontecer, pois pode ser que o Garbage Collector não tenha nem tempo de rodar antes do programa acabar.

Para a certificação, é muito importante sabermos que, por padrão na especificação, não temos como saber quando o Garbage Collector irá rodar. Porém, sabemos os momentos em que os objetos se tornam inacessíveis e consequentemente elegíveis para serem jogados fora.

Caso a prova pergunte quantos objetos foram *garbage coletados*,

deveremos responder que não sabemos dizer, mesmo tendo certeza de que há inacessíveis.

Por fim, se conseguimos referenciar indiretamente um objeto, este também não será garbage coletado. Então se tivermos uma classe chamada `Carros` que possui `c1` e `c2` do tipo `Carro`, criarmos um novo `Carros()` dentro de `TestaCicloDeVida` e passarmos `carros.c1` sendo igual a `c` e `carros.c2` igual a `c`, estaremos criando um novo objeto chamado `Carros()` que possuem duas variáveis que referenciam dois carros.

```
class Carros {
    Carro c1;
    Carro c2;
}

class TestaCicloDeVida {
    public static void main(String[] args) {

        Carros carros = new Carros();

        Carro c = new Carro();
        c.ano = 2014;
        c.modelo = "Ferrari";
        carros.c1 = c;

        c = new Carro();
        carros.c2 = c;

        c = null;

        if(15 > 10) {
```

```
            Carro c2 = new Carro();
            c2.ano = 2010;
        }

        Carro c3;
        for(int t=0;i < 10; i++) {
            c3 = new Carro();
        }
    }
}
```

COPIAR CÓDIGO

Adicionamos o primeiro e o segundo carro, então criamos objetos do tipo `Carros` na memória com as variáveis `c1` e `c2`, sem nada apontando para `null`.

Criamos um primeiro carro com o ano 2014 e modelo Ferrari e fizemos `c` apontar para ele, também fizemos com que a referência de `c1` seja a referência do mesmo objeto `c`. Depois, criamos um novo carro e pedimos com que `c` referencie este.

Com isso, nenhum objeto pode ser garbage coletado, pois o primeiro carro está sendo referenciado através da variável `carros.c1`, então ainda conseguimos acessá-lo.

Portanto, deveremos prestar atenção às referências indiretas; ainda não podem ser garbage coletados, pois podem estar sendo usados ainda.

Se a certificação perguntar quantos objetos do tipo `Carro` podem ser garbage coletados logo após a linha de `c = null`,

deveremos responder que nenhum.

Afinal, o primeiro que foi criado está sendo referenciado através de `carros.c1` e o segundo por meio de `carros.c2`. Logo, conseguiremos acessar os objetos.