02

O ciclo de vida de um objeto

O ciclo de vida dos objetos java está dividido em três fases distintas. Vamos conhecê-las e entender o que cada uma significa.

Criação de objetos

Toda vez que usamos o operador `new`, estamos criando uma nova instância de um objeto na memória:

```
class Pessoa {
    String nome;
}

class Teste {
    public static void main(String[] args) {
        Pessoa p = new Pessoa(); // criando um novo objeto do
                                 // tipo Pessoa
    }
}
```

COPIAR CÓDIGO

Repare que há uma grande diferença entre criar um objeto e declarar uma variável. A variável é apenas uma referência, um ponteiro, não contém um objeto de verdade.

```
// Apenas declarando a variável,
```

```
// nenhum objeto foi criado aqui
Pessoa p;

// Agora um objeto foi criado e atribuído a variável
p = new Pessoa();
```

COPIAR CÓDIGO

Objeto acessível

A partir do momento em que um objeto foi criado e atribuído a uma variável, dizemos que o objeto está **acessível**, ou seja, podemos usá-lo em nosso programa:

```
Pessoa p = new Pessoa(); // criação
p.nome = "Mário"; // acessando e usando o objeto
```

COPIAR CÓDIGO

Objeto inacessível

Um objeto é acessível enquanto for possível "alcançá-lo" através de alguma referência direta ou indireta. Caso não exista nenhum caminho direto ou indireto para acessar esse objeto, ele se torna **inacessível**.:

```
Pessoa p = new Pessoa();
p.nome = "Mário";

// atribuímos a p o valor null
// o objeto não está mais acessível
p = null
```

```
// criando um objeto sem variável
new Pessoa();
```

COPIAR CÓDIGO

Nesse código, criamos um objeto do tipo `Pessoa` e o atribuímos à variável `p`. Na linha **6** atribuímos `null` a `p`. O que acontece com o objeto anterior? Ele simplesmente não pode mais ser acessado por nosso programa, pois não temos nenhum ponteiro para ele. O mesmo pode ser dito do objeto criado na linha **9**. Após essa linha, não conseguimos mais acessar esse objeto.

Outra maneira de ter um objeto inacessível é quando o escopo da variável que aponta para ele termina:

```
int valor = 100;
if( valor > 50) {
    Pessoa p = new Pessoa();
    p.nome = "João";
} // Após esta linha, o objeto do tipo Pessoa não está mais
  // acessível
```

COPIAR CÓDIGO

Garbage Collector

Todo objeto inacessível é considerado elegível para o ::garbage collector::. Algumas questões da prova perguntam quantos objetos são elegíveis ao garbage collector ao final de algum trecho de código:

```
public class Bla {
```

```
    int b;
    public static void main(String[] args) {
        Bla b;
       for (int i = 0; i < 10; i++) {
            b = new Bla();
            b.b = 10;
        }
          System.out.println("fim");
    }
}
```

COPIAR CÓDIGO

Ao chegar na linha `9`, temos **9** objetos elegíveis para o Garbage Collector.

Objetos elegíveis X Objetos coletados

O ::garbage collector:: roda em segundo plano juntamente com sua aplicação java. Não é possível prever quando ele será executado, portanto não se pode dizer com certeza quantos objetos foram efetivamente coletados em um certo ponto da aplicação. O que podemos determinar é quantos objetos são elegíveis para a coleta. A prova pode tentar se aproveitar do descuido do desenvolvedor aqui: nunca temos certeza de quantos objetos passaram pelo garbage collector, logo, somente indique quantos estão passíveis de serem coletados.

Por fim, é importante ver um exemplo de referência indireta, no qual nenhum objeto pode ser "garbage coletado":

```
import java.util.*;
class Carro {

}
```

```
class Carros {
    List<Carro> carros = new ArrayList<Carro>();
}
class Teste {
    public static void main(String args[]) {
        Carros carros = new Carros();
        for(int i = 0; i < 100; i++)
            carros.carros.add(new Carro());
        // até essa linha todos ainda podem ser alcançados
    }
}
```

COPIAR CÓDIGO

Nesse código, por mais que tenhamos criados 100 carros e um objeto do tipo `Carros`, nenhum deles pode ser garbage coletado pois todos podem ser alcançados direta ou indiretamente através de nossa **thread** principal.