CURSO: Ciência da Computação	CÓDIGO: CI062
PERÍODO: 4o.	PROFESSOR: Andrey
DISCIPLINA: Técnicas Alternativas de Programação	AULA: 10

1 Apresentação

Na aula de hoje vamos apresentar e discutir e exercitar os conceitos de métodos genéricos, parâmetros de tipos e classes parametrizadas em Java. A aula de hoje é baseada em DEITEL and DEITEL (2010)

2 Desenvolvimento

Muitas vezes é necessário sobrecarregar métodos em java para tratar a mesma funcionalidade apenas com tipos diferentes. Ou então criar métodos que recebem e retornam objetos da classe Object apenas para deixá-los genéricos o suficiente para poderem ser usados em várias ocasiões. Para resolver este problema, Java implementou o conceito de parâmetros de tipos para serem usados nas definições de métodos e classes Genéricos.

A vantagem do uso de parâmetros de tipos é deixar o comportamento do método ou classe definido o suficiente para a implementação correta do método e flexível o suficiente para deixá-lo genérico.

No exemplo a seguir temos o método imprime() que é sobrecarregado para tratar tanto a impressão de um vetor de objetos Integer quanto de objetos Double.

```
1  public class ImprimeVetor {
2     public static void main(String[] args) {
3        Integer vetorInteger [] = {1,2,3,4,5,6};
4        Double vetorDouble [] = {1.0,2.0,3.0,4.0,5.0};
5        imprime( vetorInteger);
6        imprime(vetorDouble);
7     }
8     public static void imprime(Integer [] v){
9        for (Integer e: v)
10          System.out.printf("%d, ",e);
11          System.out.printf("\n");
12    }
13    public static void imprime(Double [] v){
14       for (Double e: v)
15          System.out.printf("%2.2f, ",e);
16          System.out.printf("\n");
17    }
18 }
```

2.1 Métodos Genéricos

Para não precisar sobrecarregar o método imprime() do exemplo anterior, podemos usar um método genérico com um tipo parametrizado. Para se construir um método genérico, é necessário a definição de um parâmetro de tipo com o seguinte formato: $< T >$ onde T

é um símbolo que representará o tipo a ser usado. Exemplo:

```
1   public static < T > void imprime(T [] v){
2      for (T e: v)
3         System.out.printf("%s, ",e);
4      System.out.printf("\n");
5   }
```

No exemplo acima, T será o tipo (classe) dos elementos do array a serem impressos. Fica claro que a operação a ser executada com o elemento do tipo T deve ser flexível o suficiente para servir para vários tipos, mas ela deve ter um comportamento definido. No caso do exemplo, o elemento a ser impresso é transformado em String pela opção "%s".

Para se fazer isso mais genericamente, podemos usar a definição de uma Interface.

```
1  public class Minimo {
2         public static void main(String[] args) {
3                System.out.printf("O Mínimo entre %d, %d e %d é%d \n", 3,6,4,
4                minimo(3,6,4));
5                System.out.printf("O Mínimo entre %s, %s e %s é%s\n", "tres",
6                "seis","quatro", minimo("tres","seis","quatro"));
7         }
8         public static <T extends Comparable<T>> T minimo(T a, T b, T c)
9         {
10               T menor = a;
11               if (b.compareTo(menor)<0)
12                      menor = b;
13               if (c.compareTo(menor)<0)
14                      menor = c;
15            return menor;
16        }
17 }
```

Se o tipo parametrizado for usado em vários métodos de uma classe, podemos parametrizar a própria classe. No exemplo abaixo temos uma pilha implementada com um vetor simples para armazenar objetos do tipo E.

```
1  public class Pilha< E >
2  {
3     private int tam;
4     private int top;
5     private E[] elem;
6
7     public Pilha()
8     {
9        this( 10 );
10    }
11
12    public Pilha( int s )
13    {
14       tam = s > 0 ? s : 10;
15       top = -1;
16       elem = ( E[] ) new Object[ tam ];
17    }
```

```
18
19    public void push( E v )
20    {
21       if ( top == size - 1 ) return;
22       elem[ ++top ] = v;
23    }
24
25    public E pop()
26    {
27       if ( top == -1 ) return null;
28       return elem[ top-- ];
29    }
30 }
```

3 Atividades

1: Construa uma função parametrizada que receba um vetor de elementos de um tipo e os ordene.

2: Construa uma classe parametrizada Vetor que armazene elementos de um tipo em um array e tenha métodos para ordenar o array e realizar buscas no array.

Referências

DEITEL, P. and DEITEL, H. (2010). *Java: Como programar. 8ª Edição.* Prentice Hall, São Paulo.