Texto referente às perguntas 1 e 2:

Deseja-se implementar um método *criaVec(int tam, int val)* que crie um vetor *vec* de tamanho *tam* e o inicialize com os valores {val, 2*val, 3*val, ... }.

Pergunta 1

2 pts

Assinale a alternativa que contém uma implementação correta desse método:

```
static int[] criaVec(int tam, int val){
   int[] vec = new int[tam];
   for (int i : vec) {
      vec[i] = (i+1)*val;
   }
   return vec;
}
```

```
static int[] criaVec(int tam, int val){
   int[] vec = new int[tam];
   for (int i = 0; i < vec.length; i++) {
      vec[i] = (i+1)*val;
   }
   return vec;
}
```

```
static int[] criaVec(int tam, int val){
   int[] vec = {val, 2*val, ... , tam*val};
   return vec;
}
```

```
static int[] criaVec(int tam, int val){
   int[] vec = new int[tam];
   for (int i = 0; i < vec.length; i++) {
      vec[i] = val;
      val++;
   }
   return vec;
}
```

```
static int[] criaVec(int tam, int val){
   int[] vec = new int[tam];
   for (int i = 0; i < vec.length; i++) {
      vec[i] = i*val;
      val++;
   }
   return vec;
}
```

Pergunta 2

2 pts

Assinale a alternativa que contém uma implementação possível do método main que execute *criaVec* com parâmetros tam = 10 e val = 3, e então imprima os valores do vetor resultante:

```
public static void main(String[] args) {
   int[] vec = criaVec(10, 3);
   for (int i = 0; i < vec.length; i++) {
      System.out.print(i + " ");
   }
}
```

```
public static void main(String[] args) {
   int[] vec = criaVec(tam = 10, val = 3);
   for (int i : vec) {
      System.out.print(i + " ");
   }
}
```

```
public static void main(String[] args) {
   int[] vec = new int[10];
   for (int i : criaVec(10,3)) {
      System.out.print(vec[i] + " ");
   }
}
```

```
public static void main(String[] args) {
   int[] vec = criaVec(tam = 10, val = 3);
   for (int i = 0; i < vec.length; i++) {
      System.out.print(i + " ");
   }
}
```

```
public static void main(String[] args) {
   int[] vec = criaVec(10, 3);
   for (int i : vec) {
      System.out.print(i + " ");
   }
}
```

Pergunta 3

2 pts

Deseja-se implementar um método *distancias(int[] vec)* que receba um vetor *vec* e devolva outro vetor contendo as distâncias absolutas entre cada par de valores consecutivos nesse vetor. Por exemplo, se vec = {1, 7, 3, 3, 9, -1, 0}, o método deve retornar {6, 4, 0, 6, 10, 1}, ou seja {7-1, 7-3, 3-3, 9-3, 9-(-1), e 0-(-1)). Assinale a alternativa que contém uma implementação correta desse método:

```
static int[] distancias(int[] vec){
   int[] result = new int[vec.length - 1];
   for (int i = 0; i < result.length; i++) {
      int dist = result[i+1] - result[i];
      if(dist < 0){
         dist = (-1)*dist;
      }
      result[i] = dist;
   }
   return result;
}
```

```
static int[] distancias(int[] vec){
   int[] result = new int[vec.length - 1];
   for (int i = 0; i < result.length; i++) {
      int dist = vec[i+1] - vec[i];
      if(dist < 0){
         dist = (-1)*dist;
      }
      result[i] = dist;
   }
   return result;
}
```

```
static int[] distancias(int[] vec){
   int[] result = new int[vec.length - 1];
   for (int i : vec) {
      int dist = (i+1) – (i);
      if(dist < 0){
         dist = (-1)*dist;
      }
      result[i] = dist;
   }
   return result;
}
```

```
static int[] distancias(int[] vec){
   int[] result = new int[vec.length];
   for (int i = 0; i < result.length; i++) {
      int dist = vec[i+1] - vec[i];
      result[i] = dist;
   }
   return result;
}
```

```
static void distancias(int[] vec){
   for (int i : vec) {
      int dist = vec[i+1] - vec[i];
      if(dist < 0){
         dist = (-1)*dist;
      }
      vec[i] = dist;
   }
}
```

Pergunta 4

2 pts

Deseja-se implementar um método *ehCrescente(int[] vec)* que diga se os elementos do vetor *vec* passados como parâmetro estão em ordem crescente ou não. Por exemplo, os vetores {1, 3, 3, 4, 6, 7} e {1, 1, 3, 9, 10} estão em ordem crescente, já {1, 2, 1, 2} e {1, 1, 3, 2, 10} não estão em ordem crescente. Assinale a alternativa que contém uma implementação correta desse método:

```
static boolean ehCrescente(int[] vec) {
   for (int i = 0; i < vec.length-1; i++)
      if(vec[i+1] < vec [i])
         return true;
      } else
         return false;
      }
   }
}
```

```
static boolean ehCrescente(int[] vec) {
   for (int i : vec)
      if(vec[i+1] < vec [i])
         return false;
      }
   }
   return true;
}
```

```
static boolean ehCrescente(int[] vec) {
   for (int i = 0; i < vec.length-1; i++)
      if(vec[i+1] >= vec [i]){
         return false;
      }
   }
   return true;
}
```

```
static boolean ehCrescente(int[] vec) {
   for (int i = 0; i < vec.length-1; i++) {
      if(vec[i+1] < vec [i]){
         return false;
      }
   }
   return true;
}
```

```
static boolean ehCrescente(int[] vec) {
   for (int i = 0; i < vec.length; i++) {
      if(vec[i+1] <= vec [i]){
         return false;
      }
   }
   return true;
}
```

Pergunta 5 **2 pts**

Deseja-se implementar um método *subString (char[] frase, int ini, int fim)* que retorne a parte de frase entre as posições *ini* e *fim* (inclusive) na forma de um arranjo de caracteres. Por exemplo, se frase = {'T', 'e', 's', 't', 'a', 'n', 'd', 'o'}, uma chamada a *subString(frase, 0, 3)* deve retornar {'T', 'e', 's', 't'}, enquanto uma chamada a *subString(frase, 1, 4)* deve retornar {'e', 's', 't', 'a'}. Assinale a alternativa que contém uma implementação correta desse método. Assuma que *ini* e *fim* são entradas válidas, ou seja, não é necessário testar por valores negativos ou maiores do que o tamanho da frase.

```
static char[] subString (char[] frase, int ini, int fim){
   char[] result = new char[fim-ini+1];
   for (int i : frase) {
      result[i] = frase[i];
   }
   return result;
}
```

```
static char[] subString (char[] frase, int ini, int fim){
   char[] result = new char[fim-ini+1];
   for (int i = 0; i < result.length; i++) {
      result[i] = frase[ini];
      ini++;
   }
   return result;
}
```

```
static char[] subString (char[] frase, int ini, int fim){
   char[] result = new char[fim];
   for (int i = 0; i < result.length; i++) {
      result[ini+i] = frase[fim-i];
      ini++;
   }
   return result;
}
```

```
static char[] subString (char[] frase, int ini, int fim){
   char[] result = new char[fim-ini];
   for (int i = 0; i < result.length; i++) {
      result[ini] = frase[i];
      ini++;
   }
   return result;
}
```

```
static char[] subString (char[] frase, int ini, int fim){
   char[] result = new char[fim-ini];
   for (int i = 0; i < result.length; i++) {
      result[ini] = frase[ini + i];
   }
   return result;
}
```