Deseja-se implementar um jogo de baralho, com todas as cartas incluindo coringa. Para isso, os objetos da classe Carta devem ter a seguinte especificação:

- Atributos valor, do tipo inteiro: 1 (Ás), 2, ..., 10, 11 (J: valete), 12 (Q: dama) e 13 (K: rei)
- Atributo naipe, do tipo String: “ouros”, “espadas”, “copas” ou “paus”

Para isso, é implementada a classe abaixo.

```
public class Carta {

   //Atributos estáticos (da classe)

   public static final String OUROS   = "ouros";

   public static final String ESPADAS = "espadas";

   public static final String COPAS   = "copas";

   public static final String PAUS    = "paus";


   //Atributos do objeto

   int valor;

   String naipe;


   //Construtor

   public Carta(int valor, String naipe){

      this.valor = valor;

      this.naipe = naipe;

   }


   //Método: compara com a carta “c” passada como parâmetro

   public boolean compara(Carta c){

      //Código da Pergunta 1

   }


   //Método da classe: imprime cartas em formato abreviado.

   public static void imprimeCurto(Carta[] cartas)
```

```
      //Código da Pergunta 2

   }


   //Método: busca carta no arranjo. Retorna o índice da carta se encontrada, ou -1 caso contrár
io.

   public int busca(Carta[] cartas)

      //Código das Perguntas 3 e 4

   }


   //Método: ordena carta no arranjo.

   public static void ordena(Carta[] cartas)

      //Código da Pergunta 5

   }

}
```

Pergunta 1 **2 pts**

Assinale a alternativa que indica corretamente uma possível implementação do método “compara(Carta c)” que retorna *true* apenas se dois objetos da classe Carta forem iguais:

```
public boolean compara(Carta c){

   if(valor.equals(this.c.valor) && naipe.equals(this.c.naipe))

      return true;

   return false;

}
```

```
public boolean compara(Carta c){

   if(this.valor == c.valor && this.naipe == c.naipe)

      return true;

   return false;

}
```

```
public boolean compara(Carta c){

   if(this.valor == this.c.valor && this.naipe == this.c.naipe)

      return true;

   return false;

}
```

```
public boolean compara(Carta c){

   if(this.valor.equals(c.valor) && this.naipe.equals(c.naipe))

      return true;

   return false;

}
```

```
public boolean compara(Carta c){

   if(this.valor == c.valor && this.naipe.equals(c.naipe))

      return true;

   return false;

}
```

Pergunta 2 **2 pts**

Deseja-se implementar o método "*imprimeCurto*", que recebe como parâmetro um vetor de objetos da classe Carta e imprime essas cartas em um formato abreviado contendo apenas o seu

número e a primeira letra do seu nome. Por exemplo, um Ás de ouros seria impresso como “1-o”; um 2 de espadas, como “2-e”; e um rei de copas, como “13-c”. Assinale a alternativa que implementa corretamente esse método:

```
public static void imprimeCurto(Carta[] cartas) {

   for (Carta c : cartas) {

      String s = this.valor + "-" + this.naipe.charAt(0);

      System.out.println(s);

   }

}
```

```
public static void imprimeCurto(Carta[] cartas) {

   for (Carta c : cartas) {

      String s = cartas[c.valor].valor + "-" + cartas[c.valor].naipe.charAt(0);

      System.out.println(s);

   }

}
```

```
public static void imprimeCurto(Carta[] cartas) {

   for (int i = 0; i < cartas.length; i++) {

      String s = cartas[i].toString();

      System.out.println(s);

   }

}
```

```
public static void imprimeCurto(Carta[] cartas) {

for (Carta c : cartas) {

      String s = c.valor + "-" + c.naipe.charAt(0);

      System.out.println(s);

   }

}
```

```
public static void imprimeCurto(Carta[] cartas) {

   for (Carta c : cartas) {

      String s = c.valor + "-" + c.naipe;

      System.out.println(s);

   }

}
```

Deseja-se implementar o método “busca”, por meio do qual um objeto do tipo carta procura se existe alguma carta com o mesmo valor que ela no arranjo “cartas”, passado como parâmetro (o naipe é ignorado). Considere as duas implementações desse método mostradas abaixo e assinale a alternativa correta em cada uma das questões a seguir:

```
public int buscaVersao1(Carta[] cartas) {

   for (int i = 0; i < cartas.length; i++) {

      if (this.valor == cartas[i].valor) {

         return i;

      }

   }

   return -1;

}
```

```
public int buscaVersao2(Carta[] cartas) {
```

```
   int inicio = 0;
   int fim = cartas.length - 1;
   while (inicio <= fim) {
      int meio = (inicio + fim) / 2;
      if (this.valor == cartas[meio].valor) {
         return (meio);
      }
      if (this.valor > cartas[meio].valor) {
         inicio = meio + 1;
      } else {
         fim = meio - 1;
      }
   }
   return -1;
}
```

Pergunta 3 **0.7 pts**

A implementação "*buscaVersao1*" funciona se o arranjo de entrada estiver:

- ○ Desordenado
- ◉ Ordenado ou desordenado .
- ○ Ordenado

Pergunta 4 **0.7 pts**

A implementação "*buscaVersao2*" funciona se o arranjo de entrada estiver:

- Desordenado
- Ordenado
- Ordenado ou desordenado.

Pergunta 5 **0.6 pts**

Se o arranjo de entrada estiver ordenado, qual das implementações seria mais eficiente para buscar cartas em um arranjo muito grande?

- "buscaVersao2"
- "buscaVersao1"
- Ambos teriam a mesma eficiência.

Decide-se modificar o método do Exercício 3 para que a busca não mais ignore o naipe da carta, ou seja, o valor retornado pelo método deve corresponder ao índice de uma carta com o mesmo valor e naipe do objeto que executa o método ou -1 caso não seja encontrada uma carta que satisfaça essa condição. Para isso, são sugeridas as alterações a seguir nos métodos *buscaVersao1* e *buscaVersao2*, que tiram proveito do método "compara" implementado no Exercício 1. Considere essas duas implementações e assinale a alternativa correta em cada uma das questões a seguir:

```
public int buscaVersao1(Carta[] cartas) {
   for (int i = 0; i < cartas.length; i++) {
         return i;
      }
   }
   return -1;
}
```

```
public int buscaVersao2(Carta[] cartas) {

   int inicio = 0;

   int fim = cartas.length - 1;
```

```
   while (inicio <= fim) {
      int meio = (inicio + fim) / 2;
      if (this.compara(cartas[meio])) {
         return (meio);
      }
      if (this.valor > cartas[meio].valor) {
         inicio = meio + 1;
      } else {
         fim = meio - 1;
      }
   }
   return -1;
}
```

Pergunta 6 **1 pts**

A nova implementação do "*buscaVersao1*" funciona se o arranjo de entrada estiver:

- ○ Desordenado
- ○ Nenhuma das alternativas: a implementação modificada não funciona.
- ◉ Ordenado ou desordenado .
- ○ Ordenado

Pergunta 7 **1 pts**

A implementação "*buscaVersao2*" funciona se o arranjo de entrada estiver:

- Desordenado
- Nenhuma das alternativas: a implementação modificada não funciona .
- Ordenado ou desordenado.
- Ordenado

Pergunta 8 **2 pts**

Deseja-se implementar o método da bolha para implementar o método estático "*ordena*", que ordena um arranjo de cartas de acordo com o valor de cada uma delas (ignorando o naipe). Assinale a alternativa que implementa corretamente esse método:

```
public static void ordena(Carta[] cartas) {

    int ultimo =  cartas.length - 1;

    for (int j = 0; j < ultimo; j++) {

        if (cartas[j].valor > cartas[j + 1].valor) {

            Carta aux = cartas[j];

            cartas[j] = cartas[j + 1];

            cartas[j + 1] = aux;

        }

    }

}
```

```
public static void ordena(Carta[] cartas) {

    for (int fim = cartas.length - 1; fim > 0; fim--) {

        for (int j = 0; j < fim; j++) {

            if (cartas[j].valor > cartas[j + 1].valor) {

                cartas[j] = cartas[j+1];

            }

        }

    }

}
```

```
public static void ordena(Carta[] cartas) {

    int ultimo =  cartas.length - 1;

    for (int fim = ultimo; fim > 0; fim--) {

        for (int j = ultimo; j < fim; j++) {

            if (cartas[j].valor > cartas[j + 1].valor) {

                Carta aux = cartas[j];

                cartas[j] = cartas[j + 1];

                cartas[j + 1] = aux;

            }

        }

    }

}
```

```
public static void ordena(Carta[] cartas) {

    for (int fim = cartas.length - 1; fim > 0; fim--) {

        if (cartas[fim].valor < cartas[fim-1].valor) {

            Carta aux = cartas[fim];

            cartas[fim] = cartas[fim-1];

            cartas[fim - 1] = aux;

        }

    }

}
```

```
public static void ordena(Carta[] cartas) {

for (int fim = cartas.length - 1; fim > 0; fim--) {

        for (int j = 0; j < fim; j++) {

            if (cartas[j].valor > cartas[j + 1].valor) {

                Carta aux = cartas[j];

                cartas[j] = cartas[j + 1];

                cartas[j + 1] = aux;

            }

        }

    }

}
```

Salvo em 19:46 Enviar teste