アルゴリズムとデータ構造 2016 第７回　　年　　組　　番 氏名

1. 第 4 回演習の 5.を（ヒント付きで）再掲する。下記は，最も単純な**文字列探索**アルゴリズムである。このメソッドは, 文字列 text の中に文字列 word が含まれるか先頭から調べ, 最初に見つかった位置（0 以上）を返す。ただし，見つからなかった場合には-1 を返す。空欄を適切に埋めて実行し，動作を確認せよ。

```
public static int findString(String text, String word) {

    int tlen = text.length();
    int wlen = word.length();

    // text の中の位置 t からに word が入っているか調べる（どこまで？）
    for (int t = 0; t <=          ; t++) {

        // その位置から word の文字が全部入っているか調べる（w は一致文字数）
        int w;
        for (w = 0; w <       ; w++) {

            if (text.charAt(      ) != word.charAt(    ))
                break;
        }
        if (w == wlen) return t;
    }

    return
}
```

2. 第 5 回演習の 2.と 3.を参考にして，任意のクラスの配列をそれぞれ選択ソートと挿入ソートで整列するジェネリックメソッドを作成し（※），String の配列で動作を確認せよ。

```
public static <E extends Comparable> void selectionSort(E[] a)
public static <E extends Comparable> void insertionSort(E[] a)
```

3. 第 5 回演習の 4.を参考にして，整列済みの 2 つの動的配列 ArrayList<E>（E は任意のクラス）をマージするメソッドを作成せよ。この場合，ジェネリクスの書き方がかなり複雑になり，以下のようになる。

```
// 最初の<E...>がジェネリックメソッドであることの指示（※）
public static <E extends Comparable>
ArrayList<E> merge(ArrayList<E> a, ArrayList<E> b) {

    ArrayList<E> c = new ArrayList<E>();

    int i = 0, j = 0;
    while (c.size() < a.size() + b.size()) {

        if (i >= a.size()) c.add(             );

        else if (j >= b.size()) c.add(             );

        else if (                                  ) c.add(             );

        else c.add(             );
    }
    return c;
}
```

※ <E extends Comparable> は <E extends Comparable<? super E>> と書くほうが正確である。

4. 下記のクラス Item の配列を，クイックソートで価格が**高い順**にクイックソートで整列する静的メソッド（クラスメソッド）Item.sortByPrice を作成し，別のクラスの main メソッドで使用せよ。

```
public class Item {
    public int code;   // 商品コード
    public int price;  // 価格

    public static void sortByPrice(Item[] a) {
        /* 定義を書く */
    }
}
```

5. Java API には配列操作のためのクラス java.util.Arrays があり，ソートには静的メソッド Arrays.sort（プロ II の教科書 6.5.1 を参照），2 分探索には静的メソッド Arrays.binarySearch が利用できる。API リファレンスでこれらの使用方法を調べて，動作を確認するプログラムを作成せよ。

6. 以下に示すように 4.のクラス Item を，商品コードをキーとして Arrays.sort と Arrays.binarySearch でソートおよび探索できるように修正し，配列を作って動作を検証せよ。

```
class Item implements Comparable<Item> {  // 比較可能であることを示すインタフェース

    // 4.で定義した部分はそのまま

    @Override
    public boolean equals(Object obj) { /* プロ II の教科書 14.2.6 を参考に実装 */ }

    @Override
    public int compareTo(Item item2) { /* code の小さい順になるように実装 */ }

    @Override
    public int hashCode() { return code; }  // equals をオーバーライドするときに必要
}
```

7. 通常の Arrays.sort は配列要素の compareTo メソッドを使うので，文字列をアルファベット順の逆順に並び替えたい場合には使えない。そこで，以下の順序を指定できるバージョンを使う。

```
static <E> void sort(E[] a, Comparator<? super E> comp)
```

第 2 引数には，Comparator インタフェースを実装したクラスのインスタンスを渡す。そのクラスは引数 a，b をとるメソッド compare を持ち，a を b より前に置くなら負，a と b が同じ順位なら 0，a を b より後ろに置くなら正の整数を返さなければならない。逆順に並べるためのクラスを以下に完成させよ。

```
class StringReverseComparator implements Comparator<String> {
    public int compare(String a, String b) {
        return
    }
}
```

このクラスのインスタンスを作成して Arrays.sort の第 2 引数に渡すことで（下記参照），文字列の配列をアルファベット順の逆順に並べ換える処理を実現し，動作を検証するプログラムを作成せよ。

```
Comparator<String> revcomp = new StringReverseComparator();
Arrays.sort(data, revcomp);
```