アルゴリズムとデータ構造 2017 第６回　　年　　組　　番 氏名

1. 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. 下記のクラスについて問いに答えよ（このクラスには適当なメソッドを追加してもよい）。

```
public class Item {       // 商品
    public int code;      // 品番
    public String name;   // 品名
}
```

この Item の配列に対して，挿入ソートを用いて商品を品番順に整列するメソッドと，2 分探索を用いて引数の（仮の）商品データと同じ品番の商品を探し出すメソッドを作成し，動作を検証せよ。探索で見つからなかった場合は負数を返すようにせよ。以下にメソッドの名前と引数を示す。

```
public static void sort(Item[] data)
public static int binarySearch(Item[] data, Item key)
```

3. Java ではクラスに Comparable インタフェースを実装すると比較可能（順序付け可能）になる。これによる順序を「自然な順序」という。2.の Item を以下のように品番によって順序が付くように修正せよ。

```
public class Item implements Comparable<Item> {  // Item が比較可能であることを示す

    // 2.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 this.code; }  // equals をオーバーライドするとき適切に定義
}
```

Java には配列操作のためのクラス java.util.Arrays があり，自然な順序によるソートには静的メソッド Arrays.sort（プロ II 教科書 6.5.1 参照），同じく 2 分探索には Arrays.binarySearch（各自調べること）が利用できる。修正した Item とこれらによって，2.と同じ結果を得るプログラムを作成せよ。

4. 配列を「自然な順序」以外の順序を用いて整列・探索したい場合には，まず要素同士を比較するために，java.util.Comparator インタフェースを実装した比較方法を表すクラスを作る。以下に例を示す。

```
class ItemComparator implements Comparator<Item> {  // Item のための比較方法のクラスの実装方法
    @Override
    public int compare(Item item1, Item item2) {  // 比較メソッド
        return item2.code - item1.code;  // この例では「自然な順序」の逆順の結果を返す
    }
}
```

そして，Arrays.sort(items, new ItemComparetor()) などとして，比較用クラスのインスタンスをメソッドの最後の引数として渡せばよい。この機能を利用して，Item の配列を品番ではなく，品名で並べ替え，品名で探索するプログラムを作成せよ。

5. 普通の配列は作成時のサイズ（要素数）を変更することができないが，それが不便なことも多い。そこで，Java では，java.util パッケージに ArrayList<E>（E には要素のクラス名が入る）という，サイズが動的に変更できる配列（のようなクラス）が用意されている。
   ここで，<E>は**ジェネリクス（総称型）**と呼ばれる記法で，インスタンス生成時に E の代わりに任意のクラス名を当てはめることができる。ただし，int, double などの基本型はクラスではないので，Integer, Double などのラッパークラスで代用する（プロ II 教科書 14.3.2 の autoboxing / auto-unboxing が働く）。以下に，ArrayList の使用パターンをいくつか示す。

```
import java.util.ArrayList;       // ArrayList 使用時にソースファイルの冒頭に書く

ArrayList<Integer> intArray = new ArrayList<Integer>();  // 整数の動的配列を生成
ArrayList<String> strArray = new ArrayList<String>();  // 文字列の動的配列を生成

Integer a = new Integer(32);      // 通常は Integer a = 32; と書いてよい（autoboxing）
intArray.add(a);                  // 要素を末尾に追加する（配列は自動的に拡張される）
intArray.add(64);                 // 直接 int の定数や変数を書いても自動変換される
strArray.add("abc");

int n = intArray.size();          // 動的配列のサイズ（要素数）を得る
for (int i = 0; i < n; i++) {
    int b = intArray.get(i);      // 添字 i の要素を得るには get(i)とする（auto-unboxing）
    System.out.println(b);
}

strArray.set(0, "ABC");           // strArray の 0 番目の要素"abc"を，"ABC"に入れ替える
for (String str : strArray)       // 拡張 for 文も使える
    System.out.println(str);

int j = intArray.indexOf(64)      // 先頭から線形探索でデータを探す
int k = intArray.lastIndexOf(32)  // 末尾から線形探索でデータを探す
```

これらの例を参考にして，2.で作成したプログラムを普通の配列ではなく ArrayList<Item>を利用するように書き換え，動作を検証せよ。ArrayList の詳細についてはマニュアル等を参照すること。

6. さらに，Arrays.sort を Collections.sort，Arrays.binarySearch を Collections.binarySeearch にそれぞれ書き換えると，普通の配列の代わりに ArrayList に対するソートと探索をすることができる。3.および 4.で作成したプログラムを ArrayList<Item>を利用するように書き換え，動作を検証せよ。