アルゴリズムとデータ構造 2016 第４回　　年　　組　　番 氏名

1. 下記のクラスと配列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 Member {
    public int id;       // 会員番号
    public String name;  // 会員名

    public Member(int id, String name) {
        this.id = id;
        this.name = name;
    }
}

Member [] data;  // main の中で定義
```

配列 data には会員番号順にデータを格納するものとする。この配列から会員番号をキーとして 2 分探索で会員を探し出すメソッドと，会員名をキーとして線形探索で会員を探し出すメソッドを作成し，適当なデータを設定して動作を確認せよ。なお，見つからなかった場合には，どちらも null を返すようにせよ。以下にメソッドのシグネチャ（定義の 1 行目）を示す。

```
public static Member findMemberById(int id, Member[] data)
public static Member findMemberByName(String name, Member[] data)
```

2. 普通の配列は作成時のサイズ（要素数）を変更することはできないが，配列のサイズを変更したい用途は多い。そこで，Java には ArrayList と呼ばれるサイズの変更に対応した動的な配列（のようなクラス）が用意されている。以下に ArrayList の定義から一部を抜粋した（メソッドの中身と{}は省略した)。

```
package java.util;

public class ArrayList<E> extends AbstractList<E> implements List<E>, … {
    public ArrayList()                // クラス E を要素とする配列を作る（初期サイズは 0）
    public int size()                 // 現在の要素数を返す（本物の配列の length に相当）
    public boolean add(E element)     // 配列の末尾に要素を加える（自動拡張される）
    public E get(index i)             // i 番目の要素を取り出す
    public E set(index i, E element)  // 既に存在する i 番目の要素を element に入れ替える
    …
}
```

ここで，<E>はジェネリクス（総称型）と呼ばれる記法で，インスタンスの生成時に String などの任意のクラス名に置き換えて使うことができる。ただし，int, double, float, char などの基本型はクラスではないのでそのまま適用できず，Integer, Double, Float, Character などのラッパークラスで代用する。

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

ArrayList<String> strArray = new ArrayList<String>();  // 文字列の動的配列を作成
strArray.add("abc");
strArray.add("xyz");

ArrayList<Integer> intArray = new ArrayList<Integer>();  // 整数の動的配列を作成
```

1.で作成したプログラムを，普通の配列ではなく ArrayList<Member>を利用するように書き換えよ。ArrayList の詳細についてはマニュアル等を参照すること。

3. ジェネリクスはメソッド定義の引数にも利用できる。以下の例の<E>のように，仮のクラス名を戻り値の型の前に記述すると，E を任意のクラス名に置き換えられるようになる。よってこのコードの記述だけで `String str = midElement(data, 1, 5)` というように，さまざまなクラスで利用することができる。

```
// 配列の中央の値を返す
public static <E> E midElement(E[] a, int i, int j) {
    return a[(i + j) / 2];
}
```

2 つの引数の大小関係を比較し，大きい方の値を返すジェネリックメソッド max を作成せよ。メソッドの定義の外形は以下のようになる。ただし，ここで単に<T>ではなく<T extends Comparable>としているのは，T がメソッド compareTo を持つことを保証するための記述である（※）。

```
public static <T extends Comparable> T max(T x, T y) {




}
```

※ 正確には，<T extends Comparable<? super T>>としないといけないようである。これは，Comparable もジェネリッククラスであり，さらに，T でなくても T のスーパークラス（? super T）がそのクラス用の Comparable を実装していればよいという意味だが，ここでは詳しく解説しない。

4. 第 2 回の課題の 5.で作成した線形探索メソッドを，ジェネリクスを利用することで，Object の代わりに任意のクラス T を引数とし，戻り値は T のインスタンスを返すよう書き換えよ。

```
public static <T> T linearSearch(T key, T[] data)
```

5. 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();

    for (int t = 0; t <=          ; t++) {
        int w;
        for (w = 0; w <       ; w++) {

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

    return
}
```