第九回　演習問題（標準 API）

諸注意

- 「StringUtil.java」，「Employee.java」，「EmpManager.java」の３ファイルを Web から提出する．
- コピペ発覚時は見せた側も見せてもらった側も両方０点とする．
- 必ずコンパイルエラーのない状態で提出すること（自動採点したいのでコンパイルエラーがあると，全て０点になってしまう）．
- 課題の途中で提出することになった場合，コンパイルエラーさえ出なければ，課題の途中の状態で提出してくれて構わない．一部のメソッドだけが実現できていない場合，コンパイルエラー出ないならばそのままの状態で提出してくれてよい．
- 主にコンソール出力で評価しているため，デバッグに用いたようなコンソール出力が残っていないように気をつけること．基本的にコンソール出力を指定しない限りは，課題内でコンソール出力はないものとする．
- Package は使わないこと（デフォルトパッケージで実装する）．Package で実装すると，自動採点がうまくいきません．

課題 1

<table>
<tr><td rowspan="8">1</td><td>問題設定</td><td colspan="4">　ニュース番組の原稿を作成する事になった．報道の原稿には「死語」なるものが存在し，掲載されている文言を使ってしまうと，時代遅れで視聴者が理解できず，クレームの電話につながる．挙句の果てには NEWS でキャスターが謝罪会見を開き，頭を下げる羽目になるらしい．なお謝罪会見では，司会者が途中で会見を打ち切ることになる．
　そこで事前に原稿を校正するメソッドを開発してほしい．メソッドは次の要件を確実に満たしていること．</td></tr>
<tr><td>メソッド名</td><td colspan="4">check()</td></tr>
<tr><td>引数</td><td>String text</td><td>戻り値</td><td colspan="2">String 型</td></tr>
<tr><td>処理</td><td colspan="4">　引数で送られてきた内容について，下記に示す用語集に基づいた校正を行い，放送できる内容に修正した原稿を戻り値として返す．
　はじめに private な HashMap glossary（用語集）を static フィールドとして持つ．メソッドが初めて呼び出されたときに，glossary に対して用語の追加処理を実施し，二回目以降はこれを実施せずに校正のみを行う．
　glossary は死語の用語集を管理する辞書のような役割を果たし，キーが死語，値がその置き換え用語である．すなわち，校正とはこの用語集に含まれている用語を置き換え用語に変換する処理のことを指す．</td></tr>
<tr><td>諸注意</td><td colspan="4">● static メソッドとすること．
● 全クラス，全パッケージからアクセスできるようにすること．
● StringUtil.java 内に記述すること．</td></tr>
<tr><td>放送禁止用語集
(Web 調べ)</td><td colspan="4"><table>
<tr><th>キー</th><th>値</th><th>キー</th><th>値</th></tr>
<tr><td>アベック</td><td>カップル</td><td>えもんかけ</td><td>ハンガー</td></tr>
<tr><td>チョベリバ</td><td>激おこ</td><td>井戸端会議</td><td>女子会</td></tr>
<tr><td>チョベリグ</td><td>エモい</td><td>マブダチ</td><td>ズッ友</td></tr>
<tr><td>バイビー</td><td>じゃあね</td><td>ハイカラ</td><td>オシャン</td></tr>
<tr><td>国鉄</td><td>JR</td><td>つっぱり</td><td>ヤンキー</td></tr>
</table></td></tr>
<tr><td>テスト例</td><td colspan="4">（Main.javaのmainメソッドに以下をコピペする）
<pre>String before = "昨日国鉄福井駅構内で，ハイカラなえもんかけを所持したアベックが，井戸端会議をしていた主婦らとマブダチ宣言を交わしました．";
String after = StringUtil.check(before);
System.out.println(before);
System.out.println(after);</pre></td></tr>
<tr><td>テスト出力例</td><td colspan="4">昨日国鉄福井駅構内で，ハイカラなえもんかけを所持したアベックが，井戸端会議をしていた主婦らとマブダチ宣言を交わしました．
昨日JR福井駅構内で，オシャンなハンガーを所持したカップルが，女子会をしていた主婦らとズッ友宣言を交わしました．</td></tr>
<tr><td>補足</td><td colspan="4">こんな言葉は普段から使わないほうが良い．</td></tr>
</table>

課題１：解答例（StringUtil.java）

```
// 用語集は本プログラム中に１つあれば良いのでstaticにする.
// staticにしないと，StringUtilインスタンスを作成する必要がある.
// check()内でローカルにすると，呼び出されるたびに用語集を作る手間.
private static HashMap<String, String> glossary = null;
public static String check(String text){
        // 用語集の初期化処理
        initGlossary();

        // 全てのキーバリューペアについて置換処理を実行する.
        for(Map.Entry<String, String> e : glossary.entrySet()){
                text = text.replace(e.getKey(), e.getValue());
        }
        return text;
}

// 内部処理で用いるためだけのメソッドなのでprivateにしている.
private static void initGlossary(){
        // nullチェックを行うことではじめの一回のみ初期化を行う.
        if(glossary == null){
                glossary = new HashMap<String, String>();
                glossary.put("アベック", "カップル");
                glossary.put("チョベリバ", "激おこ");
                glossary.put("チョベリグ", "エモい");
                glossary.put("バイビー", "じゃあね");
                glossary.put("国鉄", "JR");
                glossary.put("井戸端会議", "女子会");
                glossary.put("えもんかけ", "ハンガー");
                glossary.put("ナウい", "バズってる");
                glossary.put("マブダチ", "ズッ友");
                glossary.put("ハイカラ", "オシャン");
                glossary.put("つっぱり", "ヤンキー");
                glossary.put("わけわかめ", "イミぷー");
        }
}
```

課題２

<table>
<tr><td rowspan="8">2</td><td>問題設定</td><td colspan="3">部下がミスを犯して計算処理を文字列型で記述してしまった．文字列を解釈し計算結果を返すメソッドを開発してほしい．メソッドは次の要件を確実に満たしていること．</td></tr>
<tr><td>メソッド名</td><td colspan="3">calc()</td></tr>
<tr><td>引数</td><td>String formula</td><td>戻り値</td><td>double 型</td></tr>
<tr><td>処理</td><td colspan="3">引数 formula は例えば”pow(10, 2)”のような内容が送られてくる．メソッド側ではこれを，10 の 2 乗を計算するものと解釈し，100.0 を戻り値として返す．
全てに対応することは難しいため，今回は以下に示す算術メソッドについて実装を行ってほしい．それぞれに送る引数は小数の可能性もあることに注意されたい．
・pow()，abs()，log10()，max()，min()
なおそれぞれのメソッドはコンパイルエラーの書式で送られてくることはないものとして良い（すなわち，引数の数や書式は問題ないが無駄に半角スペースが挿入されているケースなどはありうる）．</td></tr>
<tr><td>諸注意</td><td colspan="3">● static メソッドとすること．
● 全クラス，全パッケージからアクセスできるようにすること．
● StringUtil.java 内に記述すること．</td></tr>
<tr><td>ヒント</td><td colspan="3">文字列を数値に変換するには各種ラッパークラスを使用すると良い．例えば int 型に変換する場合には以下のように記述する（実は第二回演習課題に一度出てきている）．
int num = Integer.parseInt(“10”);</td></tr>
<tr><td>超ヒント
（白文字）</td><td colspan="3">どうしてもアルゴリズムが思いつかない人は，以下の超（すーぱー）ヒントを一つずつ見ていくと良い（一気に読むとつまらない）．
1．
2．
3．

4．</td></tr>
<tr><td>テスト例</td><td colspan="3">（Main.javaのmainメソッドに以下をコピペする）
<pre>System.out.println(StringUtil.calc("pow(10, 2)"));
System.out.println(StringUtil.calc("abs(-1000) "));
System.out.println(StringUtil.calc(" log10(1000) "));
System.out.println(StringUtil.calc(" max(10, -2) "));
System.out.println(StringUtil.calc("min (-10 , 2) "));</pre></td></tr>
<tr><td>テスト出力例</td><td colspan="3"><pre>100.0
1000.0
3.0
10.0
-10.0</pre></td></tr>
</table>

課題２：解答例（StringUtil.java）

```
public static double calc(String formula){
        double res = 0.0d;  // 戻り値用

        // 邪魔な半角スペースを全て削除する.
        formula = formula.replace(" ", "");
        // 括弧閉じ")"は邪魔なので削除する
        formula = formula.replace(")", "");

        // ****(---,---)の書式を，****の部分と---,---の領域に分割する
        // splitは正規表現で分割するため括弧開くをエスケープする.
        // これによりsplited[0]に****が，splited[1]に---,---が格納される.
        String[] splited = formula.split("¥¥(");

        // メソッド名に応じて処理を変える
        if(splited[0].equals("pow")){
                // 2引数の場合はさらにカンマで引数を分割する.
                String[] args = splited[1].split(",");
                double a = Double.parseDouble(args[0]);
                double b = Double.parseDouble(args[1]);
                res = Math.pow(a, b);
        }else if(splited[0].equals("abs")){
                res = Math.abs(Double.parseDouble(splited[1]));
        }else if(splited[0].equals("log10")){
                res = Math.log10(Double.parseDouble(splited[1]));
        }else if(splited[0].equals("max")){
                String[] args = splited[1].split(",");
                double a = Double.parseDouble(args[0]);
                double b = Double.parseDouble(args[1]);
                res = Math.max(a, b);
        }else if(splited[0].equals("min")){
                String[] args = splited[1].split(",");
                double a = Double.parseDouble(args[0]);
                double b = Double.parseDouble(args[1]);
                res = Math.min(a, b);
        }
        return res;
}
```

課題３

3	問題設定	社員の一覧を管理するクラスを開発したい．以下の手順に従って，社員一人を管理する Employee クラスと全社員を管理する EmpManager クラスを開発せよ．

3-1	クラス名	Employee
	フィールド	name: 社員の名前(文字列型) id: 社員を一意に識別するための ID（文字列型） ＞SE001 や PG001 のように職種を示す識別子＋番号 income: 年収(数値型) birth: 生年月日(Date 型)
	定数 フィールド	定数フィールドとして public static final で宣言する． SYSTEM_ENGINEER = “SE” PROGRAMMER = “PG” HUMAN_RESOURCES = “HR” AFFAIRS = “AF”
	コンスト ラクタ	全てのフィールドを以下の引数で初期化する． name: 社員の名前(文字列型)：そのまま初期化 job: 社員の職種を示す識別子(文字列型) id: 社員 ID 用の番号(数値型) ＞job+id で SE001 のように ID フィールドを初期化 income: 社員の年収(数値型)：そのまま初期化 birth: 社員の誕生日(文字列型) ＞2018-01-01T00:00:00Z 形式か， ＞2018/01/01 00:00:00 形式で送られてくるので ＞Date 型に変換してフィールドを初期化 job が上記定数フィールド以外の値だった場合，IllegalArgumentException を発生し，エラーメッセージに「職種が誤っています．job=ジョブ」のようなメッセージを記述せよ．なお，両括弧は出力不要であり，ジョブの部分には引数として送られてきた値が入る． また誕生日の書式が異なることにより ParseException が発生した場合には，コンストラクタ内で処理せず呼び出し元に例外を投げるものとする．
	ヒント	Exception in thread "main" java.lang.IllegalArgumentException: Illegal pattern character 'T' 多分単純に実装すると上記の例外が発生する．例外の文章をよく読んでみよう．
	メソッド	String toString(): インスタンスの概要を返すメソッド 引数　：なし 戻り値：次の書式でインスタンスの概要を文字列で返す PG001_長谷川達人(4000000)_1988/12/02_12:00:00 （アンダーバーの部分は半角スペースとすること） *** getXXX(): 全フィールドの getter メソッド ＞例えば birth の場合 Date getBirth(){}となる．

課題３-１：解答例（Employee.java）

```
public class Employee {
        // 定数の定義
        public static final String SYSTEM_ENGINEER = "SE";
        public static final String PROGRAMMER = "PG";
        public static final String HUMAN_RESOURCES = "HR";
        public static final String AFFAIRS = "AF";
        // フィールドの定義
        private String name = "";
        private String id = "";   // SE001, HR001, AF001, PG001・・・
        private int income = 0;
        private Date birth;

        // birthは 2018-01-01T00:00:00Z 形式か,
        // 2018/01/01 00:00:00形式のどちらかである.
        public Employee(String name, String job, int id,
                        int income, String birth) throws ParseException{
                this.name = name;
                // 職種が誤っている場合は例外を投げる
                if(!job.equals(SYSTEM_ENGINEER)&&!job.equals(PROGRAMMER)
                && !job.equals(HUMAN_RESOURCES) && !job.equals(AFFAIRS)){
                        throw new IllegalArgumentException
                                        ("職種が誤っています. job="+job);
                }
                // 0パディングして00Xのようにするには
                // Stringのformatを使う（ググれば出てくる）.
                this.id = job + String.format("%03d", id);
                this.income = income;
                // birthにTを含む場合は邪魔な文字列を整理して書式を合わせる.
                if(birth.indexOf("T")>0){
                        birth = birth.replace("T", " ");
                        birth = birth.replace("Z", "");
                        birth = birth.replace("-", "/");
                }
                // SimpleDateFormatでDate型に変換する.
                SimpleDateFormat sdf =
                        new SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
                this.birth = sdf.parse(birth);
        }

        @Override
        public String toString(){
                SimpleDateFormat sdf =
                        new SimpleDateFormat("yyyy/MM/dd hh:mm:ss");
                return this.id + " " + this.name + "(" + this.income +
                        ") " + sdf.format(this.birth);
        }
        public String getName(){return this.name;}
        public String getId(){return this.id;}
        public int getIncome(){return this.income;}
        public Date getBirth(){return this.birth;}
}
```

3-1	テスト例	try {
		　　Employee y1 = new Employee("長谷川達人",
		Employee.PROGRAMMER, 1, 4000000, "1988-12-02T12:00:00Z");
		　　System.out.println(y1.toString());
		　　System.out.println(y1.getName());
		　　System.out.println(y1.getId());
		　　System.out.println(y1.getIncome());
		　　System.out.println(y1.getBirth());
		　　Employee y2 = new Employee("福間慎治",
		Employee.SYSTEM_ENGINEER, 1, 7000000, "1978/01/01
		12:00:00");
		　　System.out.println(y2.toString());
		　　Employee y3 = new Employee("阿笠博士", "HAKASE",
		1, 50000000, "1965/01/01 12:00:00");
		　　System.out.println(y3.toString());
		} catch (ParseException e) {
		　　e.printStackTrace();
		}

	テスト 結果例	PG001 長谷川達人(4000000) 1988/12/02 12:00:00 長谷川達人 PG001 4000000 Fri Dec 02 00:00:00 JST 1988 SE001 福間慎治(7000000) 1978/01/01 12:00:00 Exception in thread "main" java.lang.IllegalArgumentException: 職種が誤っています. job=HAKASE 　　at Employee.<init>(Employee.java:20) 　　at Main.kadai3(Main.java:51) 　　at Main.main(Main.java:11)

<table>
<tr><td rowspan="4">3-2</td><td>クラス名</td><td>EmpManager</td></tr>
<tr><td>フィールド</td><td>employees: 全社員を保持するリスト(ArrayList 型)</td></tr>
<tr><td>コンストラクタ</td><td>以下のソースコードをコピペし，社員リストに社員を登録する．
<pre>
try {
	employees.add(new Employee("サイクロン賢人",
Employee.PROGRAMMER, 1, 4000000, "1988-12-02T12:00:00Z"));
	employees.add(new Employee("ダニエル真司",
Employee.PROGRAMMER, 2, 4500000, "1986-04-11T12:00:00Z"));
	employees.add(new Employee("マンモス秀雄",
Employee.PROGRAMMER, 3, 6000000, "1970-01-01T12:00:00Z"));
	employees.add(new Employee("ジャングル隆明",
Employee.SYSTEM_ENGINEER, 1, 5000000, "1975-03-01T12:00:00Z"));
	employees.add(new Employee("レインボウ彰",
Employee.SYSTEM_ENGINEER, 2, 10000000, "1966-03-22T12:00:00Z"));
	employees.add(new Employee("マッスル晶子",
Employee.HUMAN_RESOURCES, 1, 5000000, "1982-10-30T12:00:00Z"));
	employees.add(new Employee("ケーブル妙子",
Employee.AFFAIRS, 1, 4000000, "1990-05-05T12:00:00Z"));
} catch (ParseException e) {
	// TODO 自動生成された catch ブロック
	e.printStackTrace();
}
</pre></td></tr>
<tr><td>メソッド</td><td>五種類のソートメソッドを実装する．なお実装の際は Comparator<Employee>の無名クラスを用いよ．
sortByIncome()
＞年収で昇順ソートしたリストを返す．
sortByName()
＞名前で昇順ソートしたリストを返す．
sortByBirth()
＞誕生日で昇順ソートしたリストを返す．
sortById()
＞ID で昇順ソートしたリストを返す．
sortByIdn()
＞ID の番号部分でソートし，重複している場合は職種識別子の ABC 昇順でソートしたリストを返す．</td></tr>
<tr><td>テスト例</td><td>次のページに示すのでコピペして利用すると良い．</td></tr>
</table>

課題３-２：解答例（EmpManager.java）

```
private ArrayList<Employee> employees = new ArrayList<>();
public EmpManager(){
	// 初期社員を登録する
	try {
		employees.add(new Employee("サイクロン賢人",
Employee.PROGRAMMER, 1, 4000000, "1988-12-02T12:00:00Z"));
		employees.add(new Employee("ダニエル真司",
Employee.PROGRAMMER, 2, 4500000, "1986-04-11T12:00:00Z"));
		employees.add(new Employee("マンモス秀雄",
Employee.PROGRAMMER, 3, 6000000, "1970-01-01T12:00:00Z"));
		employees.add(new Employee("ジャングル隆明",
Employee.SYSTEM_ENGINEER, 1, 5000000, "1975-03-01T12:00:00Z"));
		employees.add(new Employee("レインボウ彰",
Employee.SYSTEM_ENGINEER, 2, 10000000, "1966-03-22T12:00:00Z"));
		employees.add(new Employee("マッスル晶子",
Employee.HUMAN_RESOURCES, 1, 5000000, "1982-10-30T12:00:00Z"));
		employees.add(new Employee("ケーブル妙子",
Employee.AFFAIRS, 1, 4000000, "1990-05-05T12:00:00Z"));
	} catch (ParseException e) {
		e.printStackTrace();
	}
}
public ArrayList<Employee> sortByIncome(){
	Collections.sort(this.employees, new Comparator<Employee>(){
		@Override
		public int compare(Employee e1, Employee e2) {
			return e1.getIncome() - e2.getIncome();
		}
	});
	return this.employees;
}
public ArrayList<Employee> sortByName(){
	Collections.sort(this.employees, new Comparator<Employee>(){
		// String型はCompalableなので標準のcompareTo()が良い.
		@Override
		public int compare(Employee e1, Employee e2) {
			return e1.getName().compareTo(e2.getName());
		}
	});
	return this.employees;
}
public ArrayList<Employee> sortByBirth(){
	Collections.sort(this.employees, new Comparator<Employee>(){
		// Date型はCompalableなので標準のcompareTo()が良い.
		@Override
		public int compare(Employee e1, Employee e2) {
			return e1.getBirth().compareTo(e2.getBirth());
		}
	});
	return this.employees;
}
```

```
public ArrayList<Employee> sortById(){
	Collections.sort(this.employees, new Comparator<Employee>(){
		// String型はCompalableなので標準のcompareTo()が良い.
		@Override
		public int compare(Employee e1, Employee e2) {
			return e1.getId().compareTo(e2.getId());
		}
	});
	return this.employees;
}
public ArrayList<Employee> sortByIdn(){
	sortById();
	Collections.sort(this.employees, new Comparator<Employee>(){
		@Override
		public int compare(Employee e1, Employee e2) {
			return
e1.getId().substring(2).compareTo(e2.getId().substring(2));
		}
	});
	return this.employees;
}
```

3-2	テスト例	

```
EmpManager em = new EmpManager();
ArrayList<Employee> list = em.sortByIncome();
System.out.println("年収でソート:sortByIncome();");
for(Employee e : list){
	System.out.println(e.toString());
}
System.out.println("名前でソート:sortByName();");
list = em.sortByName();
for(Employee e : list){
	System.out.println(e.toString());
}
System.out.println("生年月日でソート:sortByBirth();");
list = em.sortByBirth();
for(Employee e : list){
	System.out.println(e.toString());
}
System.out.println("IDでソート:sortById();");
list = em.sortById();
for(Employee e : list){
	System.out.println(e.toString());
}
System.out.println("Idでソート2:sortByIdn();");
list = em.sortByIdn();
for(Employee e : list){
	System.out.println(e.toString());
}
```

3-2	テスト 結果例	年収でソート:sortByIncome(); PG001 サイクロン賢人(4000000) 1988/12/02 12:00:00 AF001 ケーブル妙子(4000000) 1990/05/05 12:00:00 PG002 ダニエル真司(4500000) 1986/04/11 12:00:00 SE001 ジャングル隆明(5000000) 1975/03/01 12:00:00 HR001 マッスル晶子(5000000) 1982/10/30 12:00:00 PG003 マンモス秀雄(6000000) 1970/01/01 12:00:00 SE002 レインボウ彰(10000000) 1966/03/22 12:00:00 名前でソート:sortByName(); AF001 ケーブル妙子(4000000) 1990/05/05 12:00:00 PG001 サイクロン賢人(4000000) 1988/12/02 12:00:00 SE001 ジャングル隆明(5000000) 1975/03/01 12:00:00 PG002 ダニエル真司(4500000) 1986/04/11 12:00:00 HR001 マッスル晶子(5000000) 1982/10/30 12:00:00 PG003 マンモス秀雄(6000000) 1970/01/01 12:00:00 SE002 レインボウ彰(10000000) 1966/03/22 12:00:00 生年月日でソート:sortByBirth(); SE002 レインボウ彰(10000000) 1966/03/22 12:00:00 PG003 マンモス秀雄(6000000) 1970/01/01 12:00:00 SE001 ジャングル隆明(5000000) 1975/03/01 12:00:00 HR001 マッスル晶子(5000000) 1982/10/30 12:00:00 PG002 ダニエル真司(4500000) 1986/04/11 12:00:00 PG001 サイクロン賢人(4000000) 1988/12/02 12:00:00 AF001 ケーブル妙子(4000000) 1990/05/05 12:00:00 IDでソート:sortById(); AF001 ケーブル妙子(4000000) 1990/05/05 12:00:00 HR001 マッスル晶子(5000000) 1982/10/30 12:00:00 PG001 サイクロン賢人(4000000) 1988/12/02 12:00:00 PG002 ダニエル真司(4500000) 1986/04/11 12:00:00 PG003 マンモス秀雄(6000000) 1970/01/01 12:00:00 SE001 ジャングル隆明(5000000) 1975/03/01 12:00:00 SE002 レインボウ彰(10000000) 1966/03/22 12:00:00 Idでソート2:sortByIdn(); AF001 ケーブル妙子(4000000) 1990/05/05 12:00:00 HR001 マッスル晶子(5000000) 1982/10/30 12:00:00 PG001 サイクロン賢人(4000000) 1988/12/02 12:00:00 SE001 ジャングル隆明(5000000) 1975/03/01 12:00:00 PG002 ダニエル真司(4500000) 1986/04/11 12:00:00 SE002 レインボウ彰(10000000) 1966/03/22 12:00:00 PG003 マンモス秀雄(6000000) 1970/01/01 12:00:00