理解度テスト　解説

1、 C,D

問題　従業員の情報を表すEmployeeクラスがある。正しくカプセル化されるようこのクラスを修正したい。修正内容として、正しいものを選びなさい。(2つ選択)

```
1 | public class Employee {
2 |     String corporateName;
3 |     String corporateAddress;
4 |     int employeeNo;
5 |     String name;
6 | }
```

☐ A.すべてのフィールドのアクセス修飾子をprivateにする。
☐ B.フィールドにアクセスするためのgetterメソッドとsetterメソッドを追加する。
☑ C.corporateNameとcorporateAddressフィールドをほかのクラスに移動する。
☑ D.名前を名乗って挨拶するメソッドを追加する。
☐ E.給与計算メソッドを追加する。

カプセル化の実現方法に関する問題です。
カプセル化は、関係するデータとそのデータを使う処理をひとまとめにする設計原則です。
カプセル化の手順は、次のとおりです。

1. 関係するデータをまとめる
2. まとめたデータを必要とする処理をまとめる

設問のEmployeeクラスはメソッドを持たないため、「関係したデータ」をまとめただけの状態です。
このクラスの名前は「従業員」を表しており、従業員に関する情報だけがまとまっていなければいけません。
しかし、Employeeクラスには、corporateNameやcorporateAddressといった「会社」に関する情報も混ざっています。
これらの情報は従業員クラスから分離し、新しく会社を表すクラスを作って移動すべきです(選択肢C)。
分離後のクラスは次のようになります。

【修正後のクラス】

このように分離することで、Employeeは従業員のデータのみ、Corporateは会社のデータのみ扱えばよくなり、クラスの役割が単純化されます。

前述のカプセル化の手順にのっとると、次はデータを必要とする処理をまとめます。
つまり、【修正後のクラス】のEmployeeクラスであれば、noとnameを使うメソッドをこのクラスに割り当てます。
選択肢Dの「名前を名乗って挨拶する」メソッドは、Employeeクラスのnameフィールドの値を必要とします。そのため、このメソッドはEmployeeクラスに割り当てなければいけません。

一方、選択肢Eの給与計算メソッドでは、働いた勤務日数や時間、雇用形態、基本給や加算給といった報酬規定の情報が必要です。しかし、これらの情報はEmployeeクラスには存在しません。
よって、この給与計算メソッドをEmployeeクラスに割り当てることはできません。
もし割り当てるなら、勤務日数や時間、雇用形態などの情報を持った勤怠クラスを新たに定義して割り当てます。

カプセル化は、データ隠蔽と組み合わせて完成します。
データ隠蔽ではフィールドのアクセス制御を行います。
よって、アクセス修飾子をprivateにするかどうかはデータ隠蔽のために必要な手段です(選択肢A)。

データ隠蔽によって、フィールドのアクセス制御が可能になり、ほかのクラスからは直接アクセスできなくなります。そこで、フィールドにアクセスするためのgetterやsetterと呼ばれるメソッドを作ることがあります。これらのメソッドは、データ隠蔽に伴って必要とされるものであり、カプセル化とは関係ありません(選択肢B)。

以上のことから、選択肢C,Dが正解です。

2、 C

問題　データ隠蔽を実現するためには、フィールドをどのように修飾すればよいか。
正しいものを選びなさい。(1つ選択)

- ☐ A.public final
- ☐ B.public static
- ☑ C.private
- ☐ D.private static
- ☐ E.private final

データ隠蔽の実現方法に関する問題です。
Javaにおけるデータ隠蔽は、アクセス制御で実現します。フィールドのアクセス修飾子をprivateにすることで、ほかのクラスからフィールド（属性）を参照できないようにします(選択肢C)。
これでほかのクラスがフィールドを直接参照すればコンパイラがエラーを出すようになります。

たとえば、次のようなSampleクラスでデータ隠蔽の結果を見てみましょう。
このクラスのフィールドnumは、privateで修飾されており、ほかのクラスからアクセスできません。

例：データ隠蔽されたSampleクラス

```
public class Sample {
    private int num = 10;
}
```

そこで、次のようにMainクラスからSampleのフィールドnumにアクセスしてみます。

例：MainクラスからSampleクラスのnumにアクセス
```
public class Main {
    public static void main(String[] args) {
        Sample s = new Sample();
        System.out.println(s.num);
    }
}
```

このMainクラスをコンパイルすると、次のようなコンパイルエラーが発生し、フィールドにアクセスできないようになっていることがわかります。このようなコンパイラによるチェックがあるおかげで、ほかのクラスからフィールドが使われていないことが保証されます。

例：MainクラスからSampleクラスのnumにアクセス
```
Main.java:4: num は Sample で private アクセスされます。
                        System.out.println(s.num);
                                           ^
エラー  1個
```

以上のことから、選択肢Cが正解です。

その他の選択肢は以下の理由により誤りです。

A.B.　フィールドのアクセス修飾子をpublicにすると「公開フィールド」という意味になるため、データ隠蔽ができません。

D.　privateなクラス変数の定義です。クラス変数にするかどうかは、カプセル化を維持することと関係はありません。

E.　finalによる定数化は、カプセル化の維持には関係ありません。

3、 B,C

問題　次のクラスにメソッドをオーバーロードして定義する場合の記述として、
正しいものを選びなさい。(2つ選択)

```
1 | public class ClassA {
2 |     int doMethod(int a) {
3 |         return 1;
4 |     }
5 | }
```

☐ A. String doMethod(int a) {
return "abc";
}
☑ B. int doMethod() {
return 1;
}
☑ C. int doMethod(int a, int b) {
return 1;
}
☐ D. int didMethod(int a) {
return 1;
}

オーバーロードメソッドの定義に関する問題です。
戻り値の型が異なっていても、シグニチャ(メソッド名と引数のセット)が同じ場合は、同一のメソッドとして扱われます。

よって、選択肢Aは誤りです。

選択肢BとCは、メソッド名は同じで引数の数が異なり、オーバーロードの要件を満たしています。
選択肢Dは、メソッド名が異なるためオーバーロードではありません。

4、 C

問題　次のコードをコンパイル、実行したときの結果として、正しいものを選びなさい。(1つ選択)

```
1  | public class Main {
2  |     public static void main(String[] args) {
3  |         Document d = new Document("James");
4  |         d.printOwner();
5  |     }
6  | }
7  | class Document {
8  |     private String owner;
9  |     public Document() {
10 |         this.owner = "none";
11 |     }
12 |     public Document(String owner) {
13 |         this.owner = owner;
14 |     }
15 |     public void printOwner() {
16 |         System.out.println(owner);
17 |     }
18 | }
```

☐ A.何も表示されない。
☐ B.「none」と表示される。
☑ C.「James」と表示される。
☐ D.コンパイルエラーになる。

オーバーロードされたコンストラクタの呼び出しに関する問題です。
オーバーロードされたメソッドを呼び出す際、引数の型や数、順番によって呼び出されるメソッドが選択されます。これはメソッドの一種であるコンストラクタも同様です。

設問のコード8～20行目のDocumentクラスには、引数なしのコンストラクタ(11～13行目)とString型の引数1つを受け取るコンストラクタ(14～16行目)が定義されています。

3行目ではDocumentクラスのインスタンスを生成しています。
コンストラクタにString型のデータを渡してインスタンス化しているので、14～16行目で定義しているString型のデータを1つ受け取るコンストラクタが呼び出されます。
コンストラクタブロック内では、引数のデータ"James"をフィールドに代入しています。
4行目でprintOwnerメソッドを呼び出し、フィールドownerが保持しているデータをコンソールに出力します。

したがって、選択肢Cが正解です。

5、 B

問題　次のコードをコンパイル、実行したときの結果として、正しいものを選びなさい。(1つ選択)

```
1  | public class Main {
2  |     public static void main(String[] args) {
3  |         ClassA a = new ClassA();
4  |         String str = a.doMethod();
5  |         System.out.println(str);
6  |     }
7  | }
8  | class ClassA {
9  |     static String doMethod() {
10 |         return "hoge";
11 |     }
12 | }
```

☐ A.何も表示されない。
☑ B.「hoge」と表示される。
☐ C.コンパイルエラーになる。

staticメソッドの呼び出しに関する問題です。
staticメソッドは、「クラス名.staticメソッド名(引数)」の書式で呼び出します。また、staticメソッドは、すべてのインスタンスで共有されるため、「インスタンスの変数名.staticメソッド名(引数)」の書式で呼び出すことも可能です。
設問のコード4行目では、インスタンスの変数名aを使って、staticメソッドを呼び出しています。
doMethod()メソッドから"hoge"が戻され、変数strに代入します。

したがって、選択肢Bが正解です。