理解度テスト　解説

1、 D

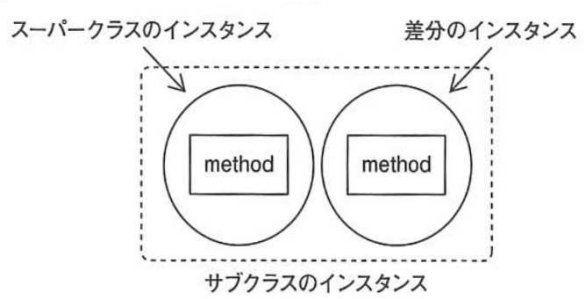
問題　サブクラスでスーパークラスのメソッドを再定義することを何と呼ぶか。
正しいものを選びなさい。(1つ選択)

- ☐ A.カプセル化
- ☐ B.ポリモーフィズム
- ☐ C.オーバーロード
- ☑ D.オーバーライド

オーバーライドに関する問題です。
オーバーライドとは、スーパークラスに定義したメソッドをサブクラスで「再定義」することです
(選択肢D)。「多重定義」を表すオーバーロードと間違えやすいので注意しましょう。

オーバーライド (override)は、メソッドの定義を上書き (overwrite)するわけではありません。
スーパークラスの定義に加えて、サブクラスに新しい定義を追加することです。
そのため、サブクラスのインスタンスには、オーバーライドされたメソッドと、オーバーライドした
メソッドの両方が含まれることになります。
たとえば、スーパークラスにあるmethodメソッドをサブクラスがオーバーライドした場合には、
次の図のようにサブクラスのインスタンスには、2つのmethodメソッドが存在します。

【オーバーライドのイメージ】



カプセル化は、関係するデータとそのデータを必要とするメソッドを1つのクラスとしてまとめること
です。メソッドの再定義とは関係ありません(部択肢A)。

ポリモーフィズムは、抽象化を実現したものです。
オーバーライドによってポリモーフィズムは実現するため、強い関係を持ちますが、再定義
そのものとは関係ありません(選択肢B)。

オーバーロードは、同じ名前で引数の種類や数、順番が異なるメソッドを複数定義すること
です。オーバーロードは「多重定義」と呼ばれます(選択肢C)。

2、 A

問題　次のAクラスのhelloメソッドをオーバーライドしたメソッド定義として、
正しいものを選びなさい。(1つ選択)

```
1 | public class A {
2 |     protected void hello() {
3 |         System.out.println("hello");
4 |     }
5 | }
```

☑ A.
```
public void hello() {
    System.out.println("B");
}
```
☐ B.
```
public String hello() {
    return "B";
}
```
☐ C.
```
public void hello(String val) {
    System.out.println(val);
}
```
☐ D.
```
void hello() {
    System.out.println("B");
}
```

メソッドをオーバーライドする条件についての問題です。
オーバーライドが成り立つ条件は、次の3点です(選択肢A)。

・メソッドのシグニチャがスーパークラスのものと同じであること
・戻り値の型がスーパークラスのメソッドと同じか、サブクラスであること
・メソッドのアクセス制御がスーパークラスと同じか緩いこと

メソッドのシグニチャが異なれば、それはオーバーロードとして見なされます。
設問のhelloメソッドは引数を受け取りませんが、選択肢CはString型の引数を受け取ります。
このようにシグニチャが異なるメソッド定義はオーバーロードです。
よって、選択肢Cは誤りです。

オーバーライドは、メソッドのシグニチャが一致しているだけでなく、戻り値の型も一致してい
なければいけません。設問のAクラスのhelloメソッドは戻り値を戻しません。
そのため、String型を戻すとした選択肢Bは誤りです。

なお、JDK5.0からはスーパークラスのメソッドで定義している戻り値型のサブクラスも戻り値型
として指定できるようになりました。このように、戻り値にサブクラスの型を指定できる機能のこと
を「共変戻り値」と呼びます。例えば、Aを継承したBというクラスがあるとき、A型を戻す次の
メソッドがあったとします。

例：共変戻り値（Bクラスで定義されているメソッド）
```
public A sample() {
    // any code
}
```

これをオーバーライドしたメソッドでは、次のようにB型を戻すことができます。

例：共変戻り値（オーバーライドしたメソッド）
```
public B sample() {
    // any code
}
```

オーバーライドの3つ目の条件は、アクセス修飾子に関するものです。ポリモーフィズムを使うためには、サブクラスは、スーパークラスができることと同じことができなくてはいけません。
もし、publicなスーパークラスのメソッドをサブクラスでオーバーライドしたときにprivateで修飾できてしまうと、オーバーライドしたメソッドをほかのクラスから呼び出せないことになり、サブクラスのインスタンスをスーパークラス型で扱うことができなくなります。
そのため、オーバーライドしたメソッドのアクセス修飾子は、スーパークラスと同じか、それよりも緩いものでなくてはいけません。

設問のhelloメソッドのアクセス修飾子はprotectedです。これより緩いものは publicだけです。
オーバーライドするメソッドのアクセス修飾子は、publicかprotectedでなくてはいけません。

したがって、選択肢Dは誤りです。

3、 B

問題　次のプログラムを実行し、画面に「sample」と表示したい。
次のプログラムの15行目にあてはまるコードを選びなさい。(1つ選択)

```
1  | public class Main {
2  |     public static void main(String[] args) {
3  |         Item item = new Item();
4  |         item.setName("sample");
5  |         System.out.println(item.getName());
6  |     }
7  | }
8  | class Item {
9  |     private String name;
10 |     public String getNae() {
11 |         return name;
12 |     }
13 |     public void setName(String name) {
14 |         // insert code here
15 |     }
16 | }
```

☐ A.name = name;
☑ B.this.name = name;
☐ C.name = this.name;
☐ D.String name = name;

変数のスコープ（有効範囲）とthisキーワードについての問題です。
メソッド内で宣言する変数のことを「ローカル変数」と呼びます。ローカル変数の名前は、命名規則にのっとっていれば自由に決められますが、宣言済みの名前は使えません。ただし、これはメソッド内に限ったことで、メソッドが異なれば同じ名前のローカル変数を宣言できます。

たとえば、次のtestメソッドでは変数aを重複して宣言しているため、3行目でコンパイルエラーが発生します。一方、6行目でも変数aを宣言していますが、これはメソッドが異なるためコンパイルエラーにはなりません。

例：同じ名前のローカル変数を重複して宣言した場合（コンパイルエラー）

```
1. public void test() {
2.     int a = 10;
3.     int a = 20;
4. }
5. public void sample() {
6.     int a = 10;
7. }
```

また、次のように引数で使っている変数名と同じ名前の変数を宣言しても、同じメソッド内で同じ名前の変数を宣言したことになるため、コンパイルエラーになります。したがって、選択肢Dは誤りです。

例：メソッドのローカル変数と同じ名前の変数を宣言（コンパイルエラー）

```
1.  public void sample(int a) {
2.      int a = 10;
3.  }
```

この「変数名は重複してはいけない」というルールはメソッド内だけに適用されるため、前述のとおりメソッドが異なる場合や、次のコードのようにフィールドとしてローカル変数が重複する場合には、コンパイルエラーは発生しません。

例：ローカル変数の宣言

```
1.  public class Sample {
2.      private int num; // フィールド
3.      public void setNum(int num) { // ローカル変数
4.          num = num;
5.      }
6.  }
```

フィールドとローカル変数の名前が同じ場合は、メソッド内でこれらの変数を使った時にはローカル変数が優先されます。そのため、上記の4行目のコードでは、ローカル変数numにローカル変数numの値を代入し直しているだけで、フィールドnumの値は変化しません。したがって、選択肢Aは誤りです。

このようにローカル変数ではなく、フィールドを使いたい場合には、thisを使います。thisは、インスタンスそのものを表すキーワードで、次のように「this.フィールド名」と記述することで、ローカル変数ではなくフィールドを使うことができます。

例：thisの使用例

```
1.  public class Sample {
2.      private int num;
3.      public void setNum(int num) {
4.          this.num = num; // フィールドnumに引数の値を代入する
5.      }
6.  }
```

以上のことから、選択肢Bが正解です。

選択肢Cでは、フィールドの値をローカル変数に代入しているだけで、フィールドの値が変化することはありません。よって、誤りです。

4、 C

問題　次のプログラムを実行し、実行結果のとおりになるようにしたい。
空欄にあてはまるコードを選びなさい。(1つ選択)

【実行結果】
super
sub

```
1 | public class SuperClass {
2 |     public void sample() {
3 |         System.out.println("super");
4 |     }
5 | }
```

```
1 | public class SubClass extends SuperClass {
2 |     public void sample() {
3 |         "                    "
4 |         System.out.println("sub");
5 |     }
6 | }
```

```
1 | public class Main {
2 |     public static void main(String[] args) {
3 |          SubClass sub = new SubClass();
4 |          sub.sample();
5 |     }
6 | }
```

☐ A.sample();
☐ B.this.sample();
☑ C.super.sample();
☐ D.SuperClass.sample();

オーバーライドしたスーパークラスのメソッド呼び出しに関する問題です。
オーバーライドはスーパークラスのメソッドを再定義するものですが、メソッドを完全に置き換えるのではなく、スーパークラスのメソッドに処理を追加したいだけのときもあります。そのようなときには、superを使ってスーパークラスのインスタンスを参照し、スーパークラスのインスタンスが持つメソッドを呼び出すことができます。

thisがそのコードを持つインスタンスそのものを表す参照であるのと同じように、superもそのコードを持つクラスのスーパークラスのインスタンスへの参照を表します。そのため、「super.メソッド名」や「super.フィールド名」という形式でスーパークラスのインスタンスのメソッドやフィールドにアクセスできます。スーパークラスのメソッドをオーバーライドし、処理を追加したい場合には、次のようにsuperを使います。

例：スーパークラスのメソッドをオーバライドする

```
public void method() {
    // 事前に追加したい処理
    super.method();
    // 事後に追加したい処理
}
```

以上のことから、選択肢Cが正解です。

選択肢AとBは同じ意味です。SubClassクラスのsampleメソッド内で同じsampleメソッドを呼び出すため、永遠に同じsampleメソッドを呼び出し続けます。このように同じメソッドを永遠に呼び出すようなコードは、次のように実行時にStackOverflowErrorが発生します。よって、選択肢AとBは誤りです。

例：選択肢AとBの実行結果

```
Exception in thread "main" java.lang.StackOverflowError
    at SubClass.sample(SubClass.java:5)
    at SubClass.sample(SubClass.java:5)
    at SubClass.sample(SubClass.java:5)
    at SubClass.sample(SubClass.java:5)
    at SubClass.sample(SubClass.java:5)
(以下省略)
```

また、選択肢Dは「クラス名.メソッド名」という形式で記述しているため、staticなメソッド呼び出しの構文になっています。しかし、SuperClassクラスのsampleメソッドはインスタンスメソッドです。そのため、この構文で呼び出すことはできません。よって、選択肢Dも誤りです。

5、 A,D

問題　次のSampleクラスを継承したサブクラスを定義するときに、
サブクラスに定義したメソッドのうち、Sampleクラスのメソッドを正しくオーバーライドしているものを選びなさい。(2つ選択)

```
1  | public class Sample {
2  |     void methodA() {}
3  |     void methodB(int a) {}
4  |     int methodC(int a, int b) {
5  |         return 0;
6  |     }
7  |     int methodD(int a) {
8  |         return 0;
9  |     }
10 | }
```

☑ A. public void methodA() {}
☐ B. public void methodB(long a) {}
☐ C. public int methodC(char a, int b) {
        return 0;
    }
☑ D. public int methodD(int i) {
        return 1;
    }

オーバライドに関する問題です。
オーバーライドとは、スーパークラスに定義したメソッドをサブクラスで「再定義」することです。
オーバーライドが成り立つ条件は、次の3点です。

・メソッドのシグニチャがスーパークラスのものと同じであること
・戻り値の型がスーパークラスのメソッドと同じか、サブクラスであること
・メソッドのアクセス制御がスーパークラスと同じか緩いこと

各選択肢については以下のとおりです。

A. シグニチャも戻り値も同じです。アクセス修飾子は異なりますが、スーパークラスのメソッドがデフォルトであるのに対し、より緩いpublicとしているため3番目のルールにも合致します。
B. スーパークラスのメソッドの引数がint型であるのに対し、long型を受け取ると定義しており、シグニチャが異なります。そのため、オーバーライドではなく、オーバーロードとして扱われます。
C. スーパークラスのメソッドとは引数の型が異なります。そのため、オーバーライドではなくオーバーロードとして扱われます。
D. シグニチャも戻り値も同じです。アクセス修飾子は異なりますが、スーパークラスのメソッドがデフォルトであるのに対し、より緩いpublicとしているため3番目のルールにも合致します。異なるのは戻す値ですが、オーバーライドかどうかはメソッドの宣言部分だけで判定し、処理内容や戻す値は考慮しません。このメソッド宣言は、オーバーライドが成立する条件を備えています。

したがって、選択肢A,Dが正解です。