理解度テスト　解説

1、 A,C

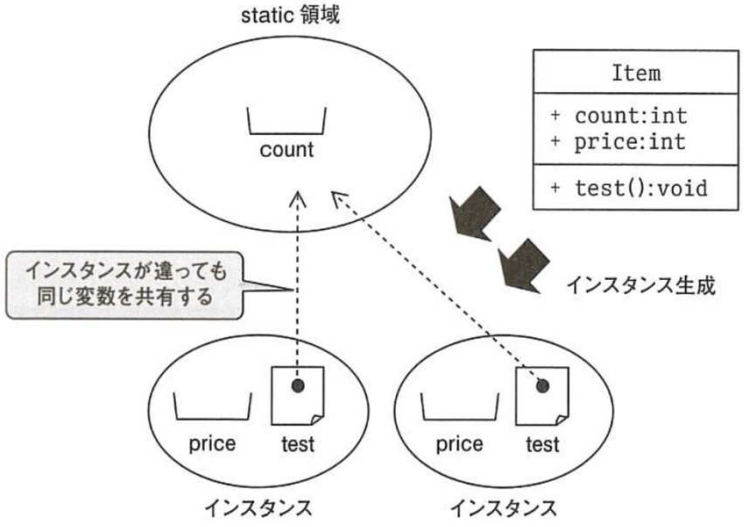
問題　次のプログラムの2行目に挿入するコードとして正しいものを選びなさい。(2つ選択)

```
1 | public class Counter {
2 |       // コードの挿入
3 | }
```

☑ A.static int count;
☐ B.int count static;
☑ C.static private int count;
☐ D.static count int;

static変数の宣言の書式に関する問題です。
static変数とは、クラスに属するフィールドのことで、「クラス変数」や「staticフィールド」とも呼ばれます。
インスタンス変数は、インスタンスごとに作られる変数だったのに対し、static変数はクラス単位で作られる変数で、次の図のようにインスタンス間で共有されることが特徴です。
static変数はインスタンスに作られる変数ではなく、static領域という場所に作られる変数です。

【static変数のイメージ】



static変数は、このように複数のインスタンスから共有されるため、あるインスタンスがstatic変数の値を変更すると、別のインスタンスからは変更後の値を参照できるようになります。
static変数の宣言は、次のように記述します。

書式　アクセス修飾子 static 型名 変数名;

宣言する場所は、クラス定義の内側、メソッドの外側です。

例：static変数の宣言

```
public class Item {
    public static int count; // static変数countの宣言
    public int price; // インスタンス変数priceの宣言
    public void test() {
        // any code
    }
}
```

各選択肢については以下のとおりです。

A. アクセス修飾子を省略し、static変数を宣言しています。
B. staticキーワードが変数宣言の最後に記述されているので誤りです。
C. staticキーワードとアクセス修飾子の記述順は逆でもかまいません。
D. 変数の型と変数名の記述順が逆になっているので誤りです。

したがって、選択肢AとCが正解です。

2、 B

問題　次のコードを実行し、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)

```
1 | public class Main {
2 |     public static void main(String[] args) {
3 |         System.out.println(CalssA.str);
4 |     }
5 | }
6 | class ClassA {
7 |     "           " String str = "hoge";
8 | }
```

【実行結果】

```
hoge
```

☐ A.void
☑ B.static
☐ C.final
☐ D.public

staticフィールドに関する問題です。
staticフィールドは、インスタンスを生成しなくても利用できるフィールドです。
staticフィールドの宣言は次にように記述します。

書式

```
アクセス修飾子 static 型名 変数名;
```

staticフィールドを宣言するには、「型名」の前にstaticキーワードを記述します。アクセス

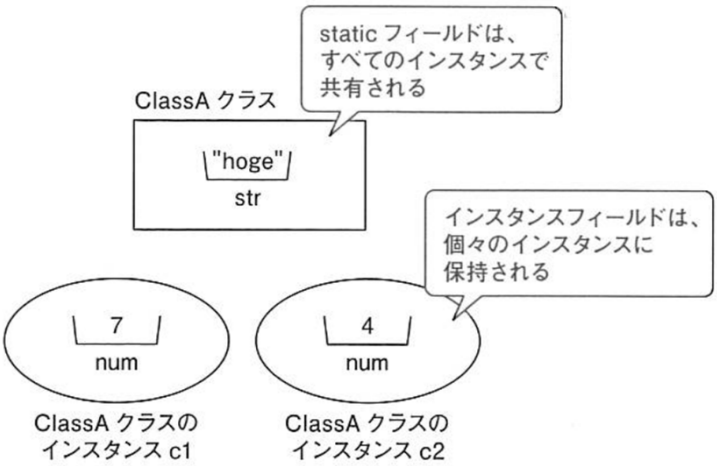
修飾子とstaticキーワードの順番を逆にしてもかまいません。

staticフィールドはインスタンス化しなくても、「クラス名.staticフィールド名」と記述して利用できます。また、staticフィールドはすべてのインスタンスで共有されます。

例：staticフィールドの宣言と利用

```
1.  public class Sample {
2.      public static void main(String[] args) {
3.          ClassA c1 = new ClassA();
4.          ClassA c2 = new ClassA();
5.
6.          ClassA.str = "hoge"; // staticフィールドの利用
7.          c1.num = 7;
8.          c2.num = 4;
9.      }
10. }
11. class ClassA {
12.     static String str; // staticフィールドの宣言
13.     int num;
14. }
```

【staticフィールドのイメージ】



設問のコードでは、インスタンス化を行わずに3行目で「クラス名.フィールド名」の形式でフィールドを利用しています。クラス名を指定してフィールドにアクセスするには、そのフィールドがstaticフィールドでなければいけません。

したがって、選択肢B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. voidメソッドの戻り値型に指定できるキーワードで、フィールド宣言には利用できません。
C. finalは定数宣言に利用するキーワードです。staticキーワードを記述しないと、インスタンスフィールドとして扱われるため、「クラス名.フィールド名」と記述できません。
D. publicキーワードはアクセス修飾子です。選択肢Cと同じく、staticキーワードを記述しないとインスタンスフィールドとして扱われます。

3、 B

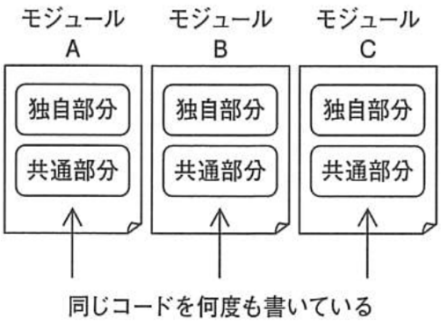
問題　Aクラスを継承したBクラスを定義しているコードとして、 正しいものを選びなさい。(1つ選択)

☐ A. public class A extends B {
　　// any code
　}
☑ B. public class B extends A {
　　// any code
　}
☐ C. public class A implements B {
　　// any code
　}
☐ D. public class B implements A {
　　// any code
　}

継承に関する問題です。
継承は、あるクラスを機能拡張した新しいクラスを定義することです。
拡張元になるクラスのことを「基底クラス」や「スーパークラス」、拡張したクラスのことを「派生クラス」や「サブクラス」と呼びます。

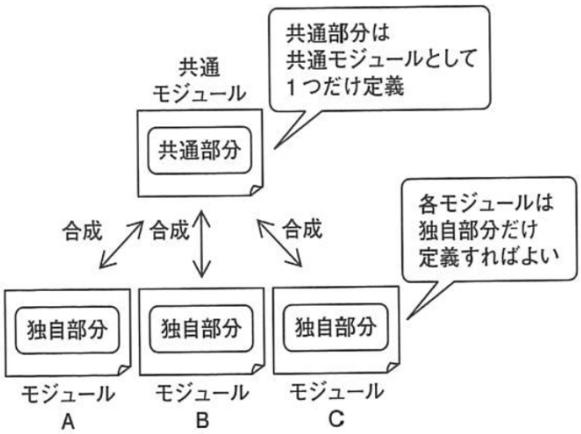
継承は、生産性を上げる非常に強力なプログラミング手法です。
複数のモジュールがあり、それぞれのコードに共通部分があったとき、同じコードを何度も記述する行為は効率的とはいえません。
もし、共通部分を持つプログラムが3つあれば、3回も同じコードを書かなければいけません。
当然、その共通部分に変更が発生すれば3つとも変更する必要があります。

【非効率なコード】



このような共通部分を別のモジュールとして分離し、それぞれのモジュールには差分だけを定義しておき、あとで結合すれば、何度も同じコードを書く手間を省くことができ、変更時に修正するコード数も減らせるというメリットもあります。
このようなプログラミング手法のことを「差分プログラミング」と呼びます。
この手法は、生産性を上げる手法として、Javaが登場する以前の1980年代から広く使われました。

【差分プログラミング】



Javaの継承を実現するには、次のようにextendsキーワードを使って、新しいクラスを定義します。

書式

```
アクセス修飾子 class クラス名 extends スーパークラス名 {
    // 拡張したい内容
}
```

たとえば、SuperClassクラスを継承したSubClassクラスは、次のように定義します。

例：クラスの継承

```
1.  public class SubClass extends SuperClass {
2.      public void sample() {
3.          // do something
4.      }
5.  }
```

extendsを訳すと「拡張する」という意味があります。
上記のクラス宣言(1行目)は、「SuperClassクラスを拡張したSubClassというクラスを定義する」と後ろから読みます。なお、sampleメソッドはSuperClassクラスにはない独自処理をするためのメソッドです。つまり、差分です。

設問では、「Aクラスを継承したBクラスを定義する」とあるため、次のようにクラスを宣言します。

例：Aクラスを継承したBクラスの定義

```
public class B extends A {
}
```

以上のことから、選択肢Bが正解となります。

選択肢Aでは「Bを継承したA」を定義してしまい、設問の逆になってしまいます。
また、選択肢CやDに使用しているimplementsは、インタフェースを実現するためのキーワードです。

継承は、同じコードをあちこちに書かずに済むため、生産性を伸ばす強力な仕組みです。
特に開発時の生産性はかなり高くなるでしょう。

4、 B,C

問題　サブクラスのインスタンスが継承元のスーパークラスから引き継げないものを選びなさい。(2つ選択)

- ☐ A.publicなメソッド
- ☑ B.privateなインスタンスフィールド
- ☑ C.コンストラクタ
- ☐ D.finalなフィールド

継承に関する問題です。
継承を使えば、スーパークラスに定義したフィールドやメソッドをサブクラスは引き継ぎます。
ただし、サブクラスはスーパークラスの定義のすべてを引き継ぐわけではありません。
サブクラスのインスタンスが、スーパークラスから引き継げないものは次の2つです。

・privateなフィールドやメソッド
・コンストラクタ

コンストラクタはサブクラスに引き継がれることはありません。
サブクラスをインスタンス化した場合、スーパークラスのインスタンスも同時に作られます。
このとき、スーパークラスのコンストラクタはスーパークラスのインスタンスの準備をしサブクラスのコンストラクタは差分のインスタンスの準備をします。コンストラクタは、そのコンストラクタが定義されているクラスのインスタンスを準備するためのメソッドなのです(選択肢C)。

アクセス修飾子privateは、同じクラスからのみアクセスを許可するアクセス修飾子です。
前述のとおり、スーパークラスのインスタンスと差分のインスタンスは異なります。
そのため、サブクラスであってもアクセスはできません(選択肢B)。

publicなメソッドは、サブクラスがオーバーライドしない限り引き継がれます(選択肢A)。
finalなフィールドは定数であり、値が変更できないだけでサブクラスのインスタンスは引き継ぎます(選択肢D)。

5、 B

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】
A
B

```
1 | public class SuperClass {
2 |     public SuperClass() {
3 |         System.out.println("A");
4 |     }
5 |     public SuperClass(String val) {
6 |         System.out.println(val);
7 |     }
8 | }
```

```
1 | public class SubClass extends SuperClass {
2 |     public SubClass() {
3 |         "                    "
4 |     }
5 |     public SubClass(String val) {
6 |         System.out.println(val);
7 |     }
8 | }
```

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

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

コンストラクタチェーンの定義についての問題です。
コンストラクタは、インスタンスの準備をするためのメソッドです。そのため、コンストラクタも通常のメソッド同様にオーバーロードできます。設問のSuperClassクラス、SubClassクラスともにコンストラクタをオーバーロードし、2つずつ定義しています。

通常のメソッドが、ほかのメソッドを呼び出せるのと同様、コンストラクタもほかのコンストラクタを呼び出せます。同じクラスに定義しているほかのコンストラクタを呼び出すには、次のようにthisを使います。
このコードでは、3行目でthisを使って5行目から始まる引数ありのコンストラクタを呼び出しています。

例：thisによるコンストラクタの呼び出し

```
1.  public class Sample {
2.      public Sample() {
3.          this("Sample");
4.      }
5.      public Sample(String val) {
6.          System.out.println(val);
7.      }
8.  }
```

また、同じクラスだけでなく、スーパークラスのコンストラクタも呼び出せます。スーパークラスのコンストラクタ呼び出しは、解答5、6で解説したとおりsuperを使います。たとえば、Aクラスを継承したBクラスのコンストラクタで、スーパークラスのコンストラクタを呼び出すには次のようにします。

例：スーパークラスのA

```
public class A {
    public A(String val) {
        System.out.println(val);
    }
}
```

例：サブクラスBからスーパークラスのコンストラクタを呼び出す

```
public class B extends A {
    public B(String val) {
        super(val);
    }
}
```

以上のとおり、選択肢AやDのようにコンストラクタ名による呼び出しはコンパイルエラーになるので誤りです。ほかのコンストラクタを呼び出すのには、thisもしくはsuperを使うことに注意しましょう。

サブクラスのコンストラクタを実行する前に、スーパークラスのコンストラクタが実行されます。
設問のSubClassクラスのコンストラクタでは、明示的にスーパークラスのコンストラクタ呼び出しをしていません。選択肢のうち、スーパークラスのコンストラクタ呼び出しをするのは選択肢Cです。
もし、選択肢Cのように引数ありのコンストラクタ呼び出しを記述すると、SubClassクラスの定義は次のようになります。

例：引数ありのコンストラクタ呼び出し

```
1.  public class SubClass extends SuperClass {
2.      public SubClass() {
3.          super("B");
4.      }
5.      public SubClass(String val) {
6.          System.out.println(val);
7.      }
8.  }
```

これでは、スーパークラスの引数ありのコンストラクタでBがコンソールに表示されるだけで、実行結果のようにA、Bが順に表示されません。そこで、次のように引数なしのコンストラクタも同時に実行できればよいのですが、スーパークラスのコンストラクタ呼び出しは1回しかできません。
そのため、次のコードは4行目でコンパイルエラーが発生します。

例：スーパークラスのコンストラクタを2回呼び出し

```
1.  public class SubClass extends SuperClass {
2.      public SubClass() {
3.          super();
4.          super("B");
5.      }
6.      public SubClass(String val) {
7.          System.out.println(val);
8.      }
9.  }
```

選択肢Bのように同じクラスのコンストラクタ呼び出しを記述すると次のようなコードになります。

例：同じクラスのコンストラクタを呼び出し

```
1.  public class SubClass extends SuperClass {
2.      public SubClass() {
3.          this("B");
4.      }
5.      public SubClass(String val) {
6.          System.out.printn(val);
7.      }
8.  }
```

サブクラスのインスタンスを作るには、必ず先にスーパークラスの準備が終わらなければいけません。
そのため、2つあるSubClassクラスのコンストラクタのどちらかで、スーパークラスのコンストラクタを呼び出さなければいけません。コンストラクタの実行は次の2つのルールに従っていなければいけません。

・スーパークラスのコンストラクタは、サブクラスのコンストラクタが実行されるより前に実行されなければいけない（ほかのコンストラクタ呼び出しは処理に含まれない）
・コンストラクタ呼び出しは、常に先頭行になければいけないため、thisとsuperを1つのコンストラクタ内で同時に実行することはできない

SubClassクラスの引数なしのコンストラクタでは、thisを使ってほかのコンストラクタを呼び出しているため、
ここにスーパークラスのコンストラクタ呼び出しを追加することは2番目のルールに違反します。
そのため、コンパイラは次のように引数ありのコンストラクタのほうに、引数なしのスーパークラスの
コンストラクタ呼び出しを追加します。

例：引数あるのコンストラクタにsuper();を追加

```
1.  public class SubClass extends SuperClass {
2.      public SubClass() {
3.          this("B");
4.      }
5.      public SubClass(String val) {
6.          super();
7.          System.out.printn(val);
8       }
9.  }
```

SubClassクラスのインスタンス化の流れをまとめると、次のようなステップで進みます。

1. SubClassクラスの引数なしのコンストラクタ呼び出し
2. SubClassクラスの引数ありのコンストラクタ呼び出し
3. SuperClassクラスの引数なしのコンストラクタ呼び出し
4. SuperClassクラスの引数ありのコンストラクタを実行し、コンソールにAが表示される
5. SubClassクラスの引数ありのコンストラクタを実行し、コンソールにBが表示される

以上のことから、選択肢Bが正解です。