理解度テスト 解説

1、 B

問題 具象クラスと抽象クラスに関する説明として、正しいものを選びなさい。(1つ選択)

- ☐ A.具象クラスは、インスタンス化できない。
- ☑ B.抽象クラスは、継承されることを前提としたクラスである。
- ☐ C.抽象クラスは、インスタンス化できる。
- ☐ D.具象クラスは、継承されることを前提としたクラスである。

具象クラスと抽象クラスの特徴に関する問題です。
クラスには、抽象クラスと具象クラスがあります。
この2つの違いは、抽象メソッドを持てるか否かです。

抽象クラスに定義する実装を持たないメソッドの宣言のことを「抽象メソッド」と呼びます。
抽象メソッドの実装は、抽象クラスを継承したサブクラスが提供します。
つまり、抽象クラスは継承を前提としたクラスであると言えます(選択肢B)。

具象クラスは、実行するためのクラスです。
そのため、全てのメソッドが実装済みでなくてはいけません。

例：具象クラス
```
public class Sample {
    public void test() {
        //do something
    }
}
```

もう一方の抽象クラスは、具象クラスとインタフェース両方の性質を併せ持っており、実装済みのメソッドに加えて、インタフェースのように抽象メソッドを定義できます。

例：抽象クラス
```
public abstract class Sample {
    public void test() {
        //do something
    }
    //実装を持たない抽象メソッド
    public abstract void hoge();
}
```

このように抽象クラスは、抽象メソッド、つまり、未実装のメソッドを持つことができるため、インスタンス化できません(選択肢C)。

具象クラスは、すべての実装を持っており、インスタンス化して利用します (選択肢A)。
具象クラスは、継承することが可能なだけであって、抽象クラスのように継承させるために存在するものではありません(選択肢D)。

2、 B,C

問題　抽象クラスに関する説明として、正しいものを選びなさい。(2つ選択)

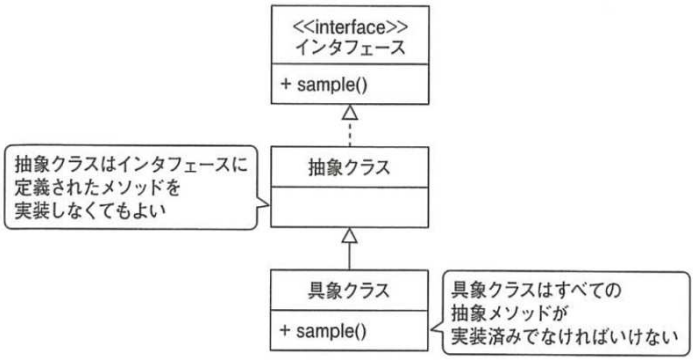
- ☐ A.すべてのメソッドは実装済みでないといけない。
- ☑ B.抽象メソッドを持つことができる。
- ☑ C.実現しているインタフェースのメソッドを実装する必要はない。
- ☐ D.インスタンスを生成できる。
- ☐ E.抽象メソッドは、暗黙的にabstractで修飾される。

抽象クラスに関する問題です。
抽象クラスは、クラスとインタフェース両方の性質を持ったクラスです。
抽象クラスと区別するために、普通のクラスのことを「具象クラス」と呼びます。
また、インタフェースに定義するメソッドを「抽象メソッド」と呼ぶのとは対照的に、具象クラスに定義するメソッドのことを「具象メソッド」と呼びます。
抽象メソッドと具象メソッドの違いは、具体的な処理内容を持つかどうかです。
クラスとインタフェース両方の性質を持った抽象クラスは、どちらのメソッドも定義できます(選択肢A、B)。

インタフェースや抽象クラスに定義した抽象メソッドは、それぞれを実現あるいは継承した具象クラスが実装しなければいけません。
ただし、このルールに従わなければいけないのは具象クラスだけです。
インタフェースがインタフェースを継承したり、インタフェースを抽象クラスが実現したり、抽象クラスが抽象クラスを継承したりした場合には、このルールは適用されません。

たとえば、次のクラス図のような関係があったとき、インタフェースに定義されている抽象メソッドsampleを実装しなければいけないのは、インタフェースを実現している抽象クラスではなくそのサブクラスである具象クラスです(選択肢C)。

【具象クラスが抽象メソッドを実装する】



抽象メソッドは、どのような処理をするべきか未定義であるため、JVMがこのようなメソッドを持つクラスを実行できません。
そのため、抽象クラスはインスタンス化できません(選択肢D)。

インタフェースには抽象メソッドしか記述できないため、コンパイラによって自動的に修飾子が追加されます。
しかし、抽象クラスには具象メソッドと抽象メソッドの両方を記述できるため、コンパイラが勝手に修飾子を追加できません(選択肢E)。
そのため、抽象クラスに定義する抽象メソッドは、プログラマーが明示的にabstractで修飾しなければコンパイルエラーになります。

3、 D

問題　インタフェースに関する説明として、誤っているものを選びなさい。(1つ選択)

☐ A.インタフェースは型を定義するためのものである。
☐ B.インタフェースには実装を定義できない。
☐ C.インタフェースはインスタンス化できない。
☑ D.インタフェースには、メソッド宣言だけが定義できる。

インタフェースの概念と特徴に関する問題です。
本設問は、情報隠蔽の実現方法として挙げたインタフェースについて問うものです。
インタフェースには、次のような特徴があります。

・実装を持たないメソッド宣言のリスト(選択肢B)
・実装を持たないため、インスタンス化できない(選択肢C)
・実装は、インタフェースを実現したクラスが提供する
・メソッドの宣言以外には、定数のみ定義できる

Javaのプログラムは、コンパイルや実行時に「型」と「実装」に分けて扱われます。
インタフェースは、この「型」を表したものです(選択肢A)。
ここでは、インタフェースを理解するために、まず「型」と「実装」が異なる概念であることを覚えましょう。

Javaのプログラミングでの基本的な単位は、「クラス」です。
このクラスには、どのように動くべきかという「実装」が記述されています。

一方、オブジェクト指向設計の基本的な単位は、「型(タイプ)」です。
型とは、オブジェクトの「種類」を表したものです。
オブジェクト指向設計では、どのような種類のオブジェクトがどのような種類のオブジェクトと連携するかを検討することから始めます。

型を使って設計すれば、過度に詳細になりがちな実装を意識しなくてもよいため、ソフトウェアの全体構造を検討できます。
そのため、オブジェクト指向設計では、全体像(ソフトウェアの構造)を検討してから、詳細な動作(実装)を検討していきます。

オブジェクト指向では、このように型と実装を異なる概念として扱います。

実際に動作するもの(実装)と、その扱い方(型)が一致する必要はありません。
そのため、ある実装を、それを持つクラスとは別の型で扱うという、ポリモーフィズムが成り立つのです。

クラスは、型と実装の両方を持っています。
実装を持たず、型だけを定義したものをインタフェースと呼びます(選択肢A)。
「実装を持たない」とは、「どのように動作すればよいかという具体的な処理を持たない」ということを意味します。
そのため、インタフェースはインスタンス化できません(選択肢C)。
実装は、インタフェースを実現したクラスが提供します。

また、型は動作しないため、変わらないものだけが定義できます。
変わらないものには、メソッドの宣言と定数があります。
具体的には、インタフェースにもメソッドの宣言とstaticな定数のみが定義できます(選択肢D)。
インタフェースは、メソッド宣言と定数のリストだと覚えておきましょう。

インタフェースとは「型」だけを定義したもので、実装は含みません。
また、実装を持たないため、インスタンス化もできません。
よって、選択肢A,B,Cは正しい説明です。

誤りを指摘する問題なので、選択肢Dが正解です。

4、 B,D,E

問題　インタフェースの特徴として、正しいものを選びなさい。(3つ選択)

- ☐ A.クラスは1つだけインタフェースを実現できる。
- ☑ B.インタフェースは、ほかのインタフェースを継承できる。
- ☐ C.インタフェースは、単一継承のみ可能である。
- ☑ D.クラスはインタフェースを多重実現できる。
- ☑ E.インタフェースには、抽象メソッドが定義できる。
- ☐ F.インタフェースには、具象メソッドが定義できる。

インタフェースの特徴に関する問題です。
インタフェースは、クラスが実現すべきメソッドの一覧を定めたものです。
インタフェースは、クラスの仕様や規約、ルールに相当するもので、具体的な処理内容については定義しません。
そのため、インタフェースには具象メソッドは定義できません(選択肢F)。
インタフェースに定義できるのは、メソッド宣言だけの抽象メソッドです(選択肢E)。

インタフェースに規定している抽象メソッドの具体的な処理は、それを実現したクラスに記述します。
インタフェースを実現するには、クラスの宣言時にimplementsを使います。
次の例では、Sampleインタフェースを実現した SampleImplクラスを定義しています。

例：インタフェースの定義
```
public interface Sample {
    void sample();
}
```

例：インタフェースを実現したクラスの定義
```
public class SampleImpl implements Sample {
    public void sample() {
        // any code
    }
}
```

クラスは、一度に複数のインタフェースを実現できます(選択肢A、D)。
複数のインタフェースを実現するには、implementsの後ろにインタフェースの名前をカンマ区切りで列挙します。
次の例では、Sampleインタフェースと Testインタフェースの2つを実現したSampleImplクラスを定義しています。

例：インタフェースを多重実現したクラスの定義
```
public class SampleImpl implements Sample, Test {
    public void sample() {
        // any code
    }
    public void test() {
        // any code
    }
}
```

インタフェースは、既存のインタフェースを継承して定義できます(選択肢 B)。
インタフェースの継承は、クラスの継承と同様にextendsキーワードを使って宣言します。
次の例では、インタフェースAを継承したインタフェースBを定義しています。

例：インタフェースAの定義
```
public interface A {
    void sample();
}
```

例：インタフェースの継承
```
public interface B extends A {
    void test();
}
```

インタフェース間の継承は、あるルールを引き継いだ(拡張した)新しいルールを規定することに相当します。そのため、インタフェースBを実現したクラスは、sampleメソッドとtestメソッドの両方を実装しなければいけません。

なお、インタフェースは、複数のインタフェースを一度に継承できます(択肢C)。
複数のものを一度に継承することを「多重継承」と呼びます。
多重継承は、インタフェースを継承するときだけ認められており、クラスの継承では認められていないことに注意しましょう。

複数のインタフェースを多重継承した新しいインタフェースは、その宣言時にextendsに続いてインタフェースをカンマ区切りで列挙して定義します。
次のコードは、インタフェースAとA2を継承した新しいインタフェースBを定義した例です。

例：インタフェースの多重継承
```
public interface B extends A, A2 {
    // any code
}
```

5、 A,B

問題　ポリモーフィズムに関係が深い用語として、もっとも適切なものを選びなさい。(2つ選択)

- ☑ A.抽象化
- ☑ B.情報隠蔽
- ☐ C.データ隠蔽
- ☐ D.カプセル化

抽象化、ポリモーフィズム、情報隠蔽の概念に関する問題です。
抽象化によって複数のモジュールをあたかも1つのモジュールであるかのように扱えます。
抽象化に使うのが「型」の情報です intやdoubleといったデータ型がそのデータの種類を表すように、型は種類であり、「同じ型」であれば「同じ種類」であることを意味します。

Javaでは、intやdoubleといったプリミティブ型以外にクラスやインタフェースも型として扱えます。
型(扱い方)と実装(動作)は異なるものです。
そのため、共通のクラスを継承したり、共通のインタフェースを実現したりすることで、異なる型のインスタンス同士でも、同じ型のインスタンスとして扱えます。
このようにインスタンスそのものの型ではなく、抽象化した型でインスタンスを扱うことを「ポリモーフィズム」と呼びます。よって、抽象化はポリモーフィズムそのものです(選択肢A)。

ソフトウェアは情報隠蔽をしなければ、改変を繰り返すたびに複雑化してしまう可能性があります。
そのため、公開する型(インタフェースやクラス)を定め、それに対応する実装(クラス)を隠蔽する情報隠蔽は、ポリモーフィズムと深い関わりを持っています(選択肢B)。

データ隠蔽は、外部のモジュールから直接内部のデータを隠蔽することです。
カプセル化に深く関わる設計原則であり、ポリモーフィズムと直接関わるものではありません(選択肢C)。

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ことを、「カプセル化」と呼びます。
カプセル化によってまとめられたモジュールをJavaでは「クラス」と呼びます。
カプセル化は、単体のクラスについての設計原則であり、ポリモーフィズムとは直接関係しません
(選択肢D)。