総仕上げ問題（Web版）解答

1. B、D

➡ Q2

インスタンスの生成に関する問題です。インスタンスを生成するには、**new**キーワードと共に、生成するインスタンスのコンストラクタを指定しなければいけません。Aクラスのインスタンスを生成するには、次のように記述します。

例 Aクラスのインスタンス生成

```
new A();
```

選択肢Bのように変数を宣言して参照を代入しても、選択肢Dのように変数を使わずにインスタンスを生成するだけでも問題ありません。以上のことから、選択肢**B**と**D**が正解です。

選択肢Aは、A型の変数を宣言しているだけで、インスタンスを生成していません。変数を宣言するだけでは、変数の中身は空であることに注意しましょう。よって、誤りです。

選択肢Cは、Bクラスのコンストラクタを呼び出しています。コンストラクタはnewキーワードと共にしか呼び出せない特殊なメソッドであるため、このコードはコンパイルエラーとなります。また、たとえnewキーワードがあったとしても生成しているのはBクラスのインスタンスで、Aクラスのインスタンスではありません。生成するインスタンスは変数の型によって変わるわけではなく、インスタンスはあくまでもコンストラクタで指定した型で生成されます。以上のことから、選択肢Cは誤りです。

クラスを継承したサブクラスをインスタンス化すると、サブクラスのインスタンスだけでなく、スーパークラスのインスタンスも生成されます。そのため、選択肢EのようにAクラスを継承したBクラスのインスタンスを生成すると、生成したBクラスのインスタンスにはAクラスのインスタンスが含まれます。しかし、Bクラスのインスタンスに含まれるAクラスのインスタンスは、Aクラスのインスタンスではなく、Bクラスのインスタンスの一部でしかありません。よって、本設問の答えとしては不適切であると言えます。したがって、選択肢Eも誤りです。

2. C

➡ Q2

拡張for文の構文に関する問題です。**拡張for文**は、カッコ「( )」の中に変数と集合（コレクションもしくは配列）をコロン「:」で区切って記述します。コロンをはさんだ左側に変数、右側に集合を記述するため、集合を右側に記述していない選択肢B、D、Fは誤りです。

選択肢Eは、コロンをはさんだ右側に配列型変数を宣言しています。コロンの右側に記述するのは、集合への参照（が代入された変数）です。よって、誤りです。

また、拡張for文内で使用する変数は、拡張for文のカッコ内で宣言しなければいけません。よって、選択肢Aは誤りです。

以上のことから、選択肢**C**が正解です。

3. C

➡ Q3

if文による条件分岐に関する問題です。if文の条件式は、boolean型の値を戻す式でなければいけません。しかし、設問のコードでは7行目で代入演算子「=」を使って変数aに変数bの値を代入しているため、int型の値が戻されます。そのため、設問のコードはコンパイルエラーとなります。以上のことから、選択肢**C**が正解です。
もし、左右オペランドの値が等しいかどうかを調べるのであれば、==演算子を使わなければいけません。

4. C

➡ Q3

二重ループの制御に関する問題です。この設問のように二重ループが出題された場合は、次のように内側のループだけを抜き出して考えましょう。

例 内側のループ

```
for (i = 1; i < 5; i++) {
    System.out.print(i);
}
```

このループを実行すると、変数iの値が1から始まり、5よりも小さい、つまり4までの間、1つずつ値を増やしながら、その値がコンソールに表示されます。そのため、コンソールには1234の数列が表示されることになります。

本設問のポイントは、使っている変数iが、内側のループ内で宣言されていな

い点です。この変数は、外側のループで宣言されているため、内側のループが終わると、変数iの値は5まで増えることになります。そのため、一度内側のループが終わり、外側のループに制御が戻ると、外側のループの条件式がfalseを戻し、外側のループを抜けてしまいます。

以上のことから、外側のループが回るのは1回だけで、コンソールには1234が1回だけ表示されます。よって、選択肢**C**が正解です。

5. B

➡ Q4

do-while文に関する問題です。while文とdo-while文は、繰り返し条件の判定を、繰り返し処理を実行する前に判定するか、それとも繰り返し処理を実行したあとに判定するかの違いがあります。

ただし、2つの違いは、初回の繰り返し処理が実行されない可能性があるときのみ生じます。**do-while文の場合は、必ず1回処理が実行されます。**while文の場合は、条件により繰り返し処理が実行されないこともあります。設問のコードでは、変数aが0から始まり、4よりも小さい間繰り返し処理が実行されます。そのため、設問のdo-while文は、次のように書き換えることができます。

例 設問のコードをwhile文に書き換え

```
while(a < 4) {
    a++;
    System.out.print("hi ");
}
```

処理の内容は、前述のとおり変数aが0から始まり、4よりも小さい間繰り返します。ただし、繰り返し処理の最初でインクリメントしているため、変数aは1から繰り返し出力されることになります。よってコンソールには1、2、3、4の順に表示され、4になった時点で条件式がfalseを戻して繰り返しを抜けます。

以上のことから、選択肢**B**が正解です。

6. D

➡ Q5

オーバーライドに関する問題です。staticメソッドはstaticメソッド同士、インスタンスメソッドはインスタンスメソッド同士でしかオーバーライドできません。
設問のコードでは、Superクラスで定義されたprintメソッドを、Subクラスでstaticで修飾してオーバーライドしています。そのため、このコードはコンパイルエラーが発生します。以上のことから、選択肢**D**が正解です。

7. B

➡ Q6

オーバーライドに関する問題です。サブクラスでスーパークラスのメソッドをオーバーライドすると、サブクラスのインスタンスを使う場合には、オーバーライドしたメソッドが実行されます。

設問のコードでは、Parentクラスを継承したChildクラスでprintInfoメソッドをオーバーライドしています。そのため、Childクラスのインスタンスを使うとChildクラスでオーバーライドしたメソッドが実行されます。よって、選択肢Aは誤りです。

設問のコードで間違えやすいのは、変数の「型」のメソッドが呼び出されるか、生成した「インスタンス」のメソッドが呼び出されるかという点です。変数の型は、インスタンスの扱い方を決めているだけであり、どのような型でインスタンスを扱おうとも、実際に動作するのはインスタンスそのものです。たとえば、設問のコードであれば、Childクラスのインスタンスを Parent型として扱っても、動作するのはChildのインスタンスです。以上のことから、ChildクラスのprintInfoメソッドが実行され、コンソールには「child」と表示されます。よって、選択肢**B**が正解です。

8. E

➡ Q7

演算子の優先順位に関する問題です。
設問のコードのポイントは4行目です。演算は左から右に順番に処理されます。そのため、まず変数bの宣言が行われます。代入演算子を実行するためには、式が処理し終わっていなければいけません。そのため、代入演算子の右オペランドが処理されます。

代入演算子の右オペランドは次のような式となっています。

例 設問のコード4行目

```
(a = 2) + a
```

この式のうち、カッコ「( )」で囲んだ式が算術演算子「+」よりも優先して処理されます。そのため、変数aに2が代入されてから、「+」演算子による演算が処理されることになります。「+」演算子が処理される段階ですでに変数aの値は2に変更されているので、2+2の結果、つまり4が変数bに代入されることになります。

以上のことから、変数aには2が、変数bには4が代入されていることになります。よって、コンソールには選択肢**E**のように「4 : 2」と表示されます。

9. C、E

➡ Q7

インタフェースのメンバについての問題です。

A. フィールドの宣言をしています。しかし、インタフェースに定義するフィールドは初期化しなければいけません。そのため、このコードはコンパイルエラーとなります。

B. メソッドの宣言をしています。インタフェースにはメソッドの実装を記述できず、宣言だけを記述します。ただし、インタフェースに定義できるメソッドの宣言は、publicなものだけです。このメソッド宣言はprivateで修飾しているので、コンパイルエラーとなります。

C. このメソッド宣言は、publicで修飾されていません。このように明示的にpublicを記述しなかったとしても、コンパイラによって自動的にpublicが追加されるためコンパイル可能です。

D. staticなメソッドの宣言です。staticメソッドの宣言は、インタフェースに定義できません。よって、コンパイルエラーとなります。

E. publicなメソッドの宣言です。このコードは、メソッドの実装を持たない宣言のみの定義です。このメソッドはインタフェースに定義できるコンパイル可能なコードです。

したがって、選択肢**C**と**E**が正解です。

10. A、B、D

➡ Q8

クラスの宣言方法についての問題です。

選択肢Aは、もっとも一般的なクラス宣言です。よって、選択肢**A**は有効なクラス宣言です。

選択肢Bは、java.lang.Objectクラスを継承している点とアクセス修飾子がデフォルトになっている点が選択肢Aと異なります。すべてのクラスはjava.lang.Objectクラスのサブクラスとなることが言語仕様で決まっています。選択肢Aのようにプログラマーが明示的に継承を記述しなかった場合でも、コンパイラが自動的にjava.lang.Objectクラスを継承するコードを追加します。また、クラス宣言時に使えるアクセス修飾子は、publicとデフォルトの2種類です（インナークラスの場合は、これら以外の修飾子も使えます）。以上のことから、選択肢**B**は有効なクラス宣言です。

選択肢Cは、継承の宣言時にアスタリスク「*」を使ってワイルドカード指定をしています。継承は、必ずどのクラスを継承するのかを明示しなければいけません。よって、このようにワイルドカードを使って、java.langパッケージのいずれかのクラスを継承するというような宣言はできません。よって、選択肢Cは無効なクラス宣言です。

選択肢Dは、finalクラスの宣言です。finalクラスはサブクラスを作ることができないクラスのことです。たとえばjava.lang.Stringクラスはfinalクラスの一例です。よって、選択肢**D**は有効なクラス宣言です。

選択肢Eは、Objectクラスをimplementsしようとしています。implementsは、インタフェースを実現するときに使うキーワードで、クラスに対して使うことはできません。よって、選択肢Eは無効なクラス宣言です。

11. B

➡ Q8

サブクラスのインスタンスからスーパークラスのインスタンスにアクセスするための方法に関する問題です。サブクラスをインスタンス化すると、サブクラスとスーパークラスの両方のインスタンスが生成され、両方で1つのインスタンスとして扱われます。そのため、サブクラスのインスタンスからスーパークラスのインスタンスにアクセスすることが可能です。

サブクラスのインスタンスからスーパークラスのインスタンスにアクセスするタイミングは次の2とおりです。

- サブクラスのコンストラクタ内でスーパークラスのコンストラクタを呼び出す
- サブクラスのメソッド内でスーパークラスのメソッドやフィールドにアクセスする

1つ目のタイミングでは、スーパークラスのコンストラクタを**super()**で呼び出します。2つ目のタイミングでは、**super**を使ってアクセスします。

設問のコードで「B, A」と表示するには、サブクラスのprintメソッド内で、スーパークラスのフィールドにアクセスしなければいけません。そのため、選択肢Bのようにsuperを使ってアクセスします。選択肢Dは、スーパークラスのコンストラクタを呼び出しています。以上のことから、選択肢**B**が正解で、選択肢Dは誤りです。

選択肢Aは、インスタンス自身を表すthisを使っています。設問のコードでは、thisはBクラスのインスタンスを表すことになります。そのため、コンソールには「B, B」と表示されます。よって、選択肢Aは誤りです。

選択肢Cは、コンストラクタ内で同じクラスに定義されている別のコンストラクタを呼び出すコードです。このようなコンストラクタを呼び出すコードは、コンストラクタ内でしか呼び出せないため、コンパイルエラーとなります。よって、選択肢Cも誤りです。

選択肢Eのような「**クラス名.フィールド名**」という書式は、staticフィールドにアクセスするときの記述方法です。しかし、設問のAクラスに定義されたnameフィールドはインスタンスフィールドです。よって、このコードはコンパイルエラーが発生します。以上のことから、選択肢Eは誤りです。

12. C ➡ Q9

インクリメント演算子の前置と後置の違いに関する問題です。**インクリメント演算子**が**前置の場合は代入の前**に、**後置の場合は代入後**に値が増えると覚えておくとよいでしょう。

設問のコードでは、4行目で変数num2にインクリメントされたnum1の値を代入しています。このとき、インクリメント演算子は後置されているため、変数num2にはインクリメントされる前の値、つまり4が代入されます。代入後、変数num1の値は1増えて4から5に変わります。

次に5行目では、変数num3にインクリメントされたnum2の値を代入しています。このときは、インクリメント演算子が前置されているため、変数num3に値を代入する前に、num2の値が4から5に増えます。その後、num2の値がnum3に代入されるため、num2にもnum3にも5が代入されていることになります。

以上のことから、num1、num2、num3のすべての変数に5が代入されていることがわかります。よって、選択肢**C**が正解です。

13. B ➡ Q10

オーバーライドとポリモーフィズムに関する問題です。サブクラスのインスタンスをスーパークラス型として扱っても、動作するのはサブクラスのインスタンスです。変数の型は、あくまでも扱い方を宣言するもので、インスタンスの型を変更するわけではありません。そのため、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場合には、変数がスーパークラス型であろうとオーバーライドしたメソッドが有効になります。

設問のコードでは、Aクラスを継承したBクラスを定義し、さらにtestメソッドをオーバーライドしています。そのため、Bクラスのインスタンスのtestメソッドを呼び出すと、A型で扱っていようとオーバーライドしたメソッドが

実行されます。よって、コンソールには「B」が表示されます。以上のことから、選択肢**B**が正解です。

14. B

➡ Q11

switch文の制御に関する問題です。switch文の主な出題ポイントは、**break文**による制御です。switch文では、式の値がcaseラベルで指定した値に一致した箇所からbreak文が表れるまで、すべての処理（ほかのcaseラベルとdefaultラベルの処理も含む）が実行されます。どこにbreak文が使われていて、どこに使われていないかを確認するようにしましょう。

設問のコードでは、変数numの値が1なので、1つ目のcaseラベルに一致します。しかし、このcaseラベルに対応する処理にはbreak文が表れないため、コンソールに「A」が表示されたあと、そのまま次のcaseラベルの処理も実行してしまいます。そのため、コンソールには「A」に続いて「B」が表示され、その後、break文によってswitch文のブロックから抜けます。

以上のことから、選択肢**B**が正解です。

15. C

➡ Q11

Javaのエディションに関する問題です。Javaには、Java SE、Java EE、Java MEの3つのエディションがあります。選択肢DのJava DBは、Javaが提供する簡易DBMS（データベース管理システム）であり、エディションの名前ではありません。よって、誤りです。

選択肢Aの**Java SE**は、ほかの2つのエディションの基となり、Javaの基本的な機能を提供するエディションです。実際の業務アプリケーションなどでは、Java SEを応用したJava EEやJava MEが使われます。Webベースのシステムを開発するには、**Java EE**を使うのが一般的です。Java SEは基本機能を提供しているため、Java SE単独でもWebベースのシステムを開発できないことはありませんが、非効率的すぎるため適していません。以上のことから、選択肢**C**が最適と言えます。

選択肢Bの**Java ME**は「マイクロエディション」と呼ばれ、ロボットなどの産業機器、モバイル機器で使うためのエディションです。よって、誤りです。

16. C

➡ Q12

ローカル変数の初期化に関する問題です。初期化されていないローカル変数は参照できません。もし、初期化されていないローカル変数を参照するようなコードを記述するとコンパイルエラーとなります。一方、インスタンスの

フィールドとして宣言した変数は、自動的にデフォルト値で初期化されます。そのため、フィールドの場合には初期化しなくてもコンパイルエラーとはなりません。間違えやすいので注意しましょう。

設問のコードでは、mainメソッド内で2つのSample型変数s1とs2を宣言しています。しかし、これらの変数が初期化されることなく、次の行でnameフィールドに値を代入しようとしています。そのため、このコードはコンパイルエラーが発生します。以上のことから、選択肢**C**が正解です。

17. C

➡ Q13

後置デクリメントの実行タイミングに関する問題です。**デクリメント演算子**も、前置と後置で実行のタイミングが異なります。気を付けなければいけないのは、後置されたときです。**後置の場合は、処理が実行されてからデクリメントが実行される**ことに注意しましょう。

設問のコードでは、変数iが3で初期化され、デクリメントされながら表示されていきます。このとき、デクリメント演算子は後置されているため、値は表示されてから1減ります。よって、最初に表示されるのは3です。以上のことから、数列が2から始まる選択肢Aは誤りです。

while文の条件式は、iの値が0以上であればtrueを戻すため、3、2、1、0と順に表示し、その後、iの値が-1になった時点でwhile文から抜けます。以上のことから、選択肢**C**が正解です。

18. B

➡ Q13

データ型の範囲に関する問題です。すべてのデータ型の範囲を正確に覚える必要はありませんが、範囲が狭いbyte型とshort型の2つについては覚えておきましょう。

byte型は-128～127まで、short型は-32768～32767までの値を表すことができます。覚えられない場合には、byte型が8ビット、short型はその倍と覚えておいて、次のように2進数で表してみるとよいでしょう。

【byte型とshort型の範囲】

byte									0	0	0	0	0	0	0	0
short	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10 進数	32768	16394	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

このように2進数で0、1がいくつ並ぶかを考え、10進数に直すとちょうど最後のビットが表す値を倍にすると、その型が表せる数値の範囲となり、その半

分が正の値、もう半分が負の値となります。
たとえば8ビットのbyte型の場合、2進数で表す値の最小が「00000000」、最大が「11111111」です。最大「11111111」のとき、最後の8ビット目が表すのは128という値です。つまり、「10000000」は128を表しています。この最後のビットが表す数を倍にした256がbyte型が表せる数字の範囲であり、byteは256通りの数値を表すことができます。byte型は正の値と負の値の両方を表すことができるため、256通りのうち半分を正の値（0〜127）、残り半分を負の値（-128〜-1）のために使います。正の値は0を含んで128通りなのがポイントです。

こうすれば、細かな値を覚える必要はありませんので、迷ったらこのような表を書いてみましょう。

設問のコードでは、選択肢Bがshort型の変数にその範囲を超える値70000を代入しています。この値はint型、もしくはlong型の変数でしか扱えません。よって、選択肢**B**がコンパイルエラーとなるコードです。

19. B、E、G ➡ Q13

Javaの特徴に関する問題です。

A. Javaは、コンパイラによってJVMが解釈しやすい形に変更されます。よって、Javaの特徴を正しく述べています。

B. Javaは、JVMがプラットフォームごとの違いを吸収するため、JVMさえ用意できれば、どのようなプラットフォームでも動作可能です。Javaのプログラムは「Write once, Run anywhere」の標語のとおり、WindowsでもMac OSでもUnixでも動作できることが大きな特徴となっています。よって、説明は誤りです。

C. Javaは、Java EEというエディションで、分散プログラミングや分散コンピューティングを標準でサポートしています。Java EEを使えば、複数の物理的に異なるコンピュータ同士を連携させ、あたかも1つのコンピュータで動作しているかのようなアプリケーションを開発できます。よって、正しい説明です。

D. Javaは、さまざまなハードウェア上で動作可能です。個人向けのパーソナルコンピュータはもちろんのこと、複数のユーザーが同時にアクセスするサーバー、モバイル機器、ロボットの制御など、さまざまな環境で動作します。よって、正しい説明です。

E. Javaは、標準でマルチスレッドによる並行処理をサポートしています。従

来、並行処理はたいへんなプログラミングでしたが、Javaは簡単に並行処理のプログラミングができるようになっています。よって、説明は誤りです。

F. JavaのJVMには、「ガベージコレクタ」と呼ばれる機能が実装されており、プログラムの実行中に消費したメモリのうち、不要になったものを自動的に解放することで、メモリを有効に使える機能が搭載されています。そのため、Javaを使ったプログラミングでは、煩わしいメモリ管理のコードをプログラマーが記述する必要がありません。よって、正しい説明です。

G. Javaには自動メモリ管理機能が標準で搭載されている一方、プログラマーがポインタを使って自由にメモリ操作を行うことはできません。よって、説明は誤りです。

誤りを指摘する問題なので、選択肢**B**、**E**、**G**が正解です。

20. B、C、E

➡ Q14

アクセス制御の範囲に関する問題です。

クラスの**コンストラクタ**は、オーバーロードして複数定義することが可能です。そのとき、外部のクラスに公開したいコンストラクタと公開したくないコンストラクタに分けることができます。公開したいコンストラクタには**public**を、公開したくない、もしくは公開範囲を限定したいコンストラクタにはそれ以外の修飾子を付けることができます。もちろん、privateで修飾することも可能です（選択肢A）。
なお、**private**で修飾されたコンストラクタは、外部のクラスからは使えませんが、同じクラスに定義されている別のコンストラクタからthis()を使って呼び出すことが可能です。

インタフェースは、外部に公開する宣言部分だけを抜き出したものです。そのため、privateを使ってメソッドを非公開にすることはできません。同様にインタフェースに定義する定数も公開するためのものです。よって、定数も非公開にすることはできません（選択肢**B**、選択肢**E**）。

抽象メソッドは、サブクラスでの実装を強制するためのメソッドです。privateで修飾するとサブクラスが継承して実装を提供できなくなってしまいます（選択肢**C**）。

具象メソッドは、実装内容を持たない抽象メソッドとは反対に、実装内容を持つメソッドです。外部のクラスに非公開にしたいメソッドはprivateで修飾します（選択肢D）。

クラスの変数は、何か設計上の理由がない限り、データ隠蔽の原則に従って、ほかのクラスから直接変更されたりしないようにprivateで修飾するのが一般的です（選択肢F）。

21. B

➡ Q15

サブクラスのコンストラクタ内でのスーパークラスのコンストラクタ呼び出しに関する問題です。

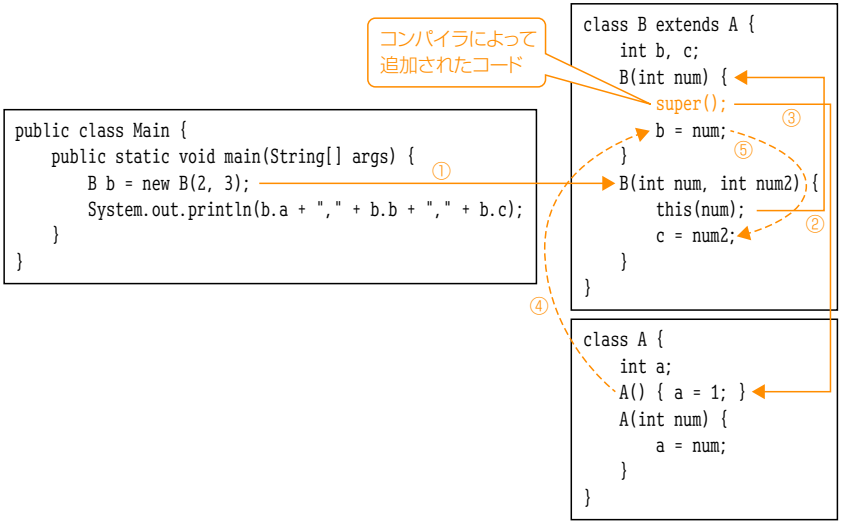
コンストラクタは、インスタンスを利用する前の準備をする特別なメソッドです。2つのクラスに継承関係が成り立つとき、サブクラスのインスタンスを準備するより前に、スーパークラスのインスタンスの準備を整えなければいけません。そのため、スーパークラスのコンストラクタを実行したあとに、サブクラスのコンストラクタを実行する必要があります。このようなルールを実現するために、サブクラスのコンストラクタの先頭行では、スーパークラスのコンストラクタ呼び出しをしなければいけないことが言語仕様で決められています。

設問のコードでは、Mainクラスの3行目で、int型の引数を2つ受け取るコンストラクタを使ってBクラスのインスタンスを生成しています。このコンストラクタでは、**this**を使ってオーバーロードされた別のコンストラクタを呼び出しています。サブクラスのコンストラクタ内でのスーパークラスのコンストラクタ呼び出しよりも先に何らかの処理を記述することはできませんが、オーバーロードされた別のコンストラクタを呼び出すコードは記述できます。その場合は、オーバーロードされた別のコンストラクタの先頭行で、スーパークラスのコンストラクタ呼び出しをする必要があります。

このスーパークラスのコンストラクタ呼び出しのコードは、プログラマーが明示的に記述しなかった場合、コンパイラが自動的にコードを追加します。そのため、コンパイル後のコンストラクタの定義は次のようになります。

例 コンパイラによって自動的にコードが追加される

```
B(int num) {
    super();   ← コンパイラによって追加されたコード
    b = num;
}
B(int num, int num2) {
    this(num);
    c = num2;
}
```



Bクラスのスーパークラスであるクラスの引数なしのコンストラクタでは、変数aに1を代入しています。また、Bクラスのint型の引数を1つだけ受け取るコンストラクタでは変数bに2を、引数を2つ受け取るコンストラクタでは変数cに3を代入しています。そのため、コンソールには、「1,2,3」が表示されます。以上のことから、選択肢**B**が正解です。

22. A

➡ Q15

アクセス修飾子に関する問題です。

privateで修飾されたメソッドは、継承関係の有無にかかわらず、同じクラスからしかアクセスできません。また、同じパッケージに属していても、privateメソッドにはアクセスできません。以上のことから、選択肢**A**が正解で、選択肢BとCは不正解です。

サブクラスからスーパークラスのメンバにはアクセスができますが、スーパークラスからサブクラスのメンバにはアクセスできません。よって、選択肢Dはアクセス修飾子の種類にかかわらず誤りです。

23. B

➡ Q16

staticメソッドの特徴に関する問題です。

staticメソッドは、インスタンスを生成しなくても使えるメソッドです。一方、staticが付いていないメソッド（インスタンスメソッド）は、インスタンスを生成しなければ使えません。Javaでは、staticメソッドからはstaticメソッドにしかアクセスできません。これは、インスタンスの生成後に、同一クラス内のstaticメソッド（インスタンスを生成しなくても呼び出し可能）から、staticが付いていないメソッドを呼び出せてしまうと、インスタンスが存在しないのに、staticが付いていないメソッドが呼び出されてしまうという矛盾を生じさせないためです。
よって、設問のコードでは、staticメソッドであるmainメソッドからアクセスするために、halloメソッドはstaticで修飾されている必要があります。したがって、選択肢**B**が正解です。

24. D

➡ Q16

アクセス制御と継承に関する問題です。

Aクラスのprivateなnumフィールドにアクセスしているのは、同じクラスに定義されているprintメソッドです。privateは異なるクラスからのアクセスを禁じていますが、同じクラスのメソッドであればアクセスできます。よって、選択肢Aは誤りです。

サブクラスは、スーパークラスに定義されているprivate以外のフィールドやメソッドを引き継ぎます。そのため、設問のAクラスを継承したBクラスはnumフィールドを引き継ぎませんが、printメソッドは引き継ぎます。Bクラスの4行目に記述されているprintメソッドの呼び出しは正しいコードです。よって、選択肢Bは誤りです。

2つのクラスが継承関係にあるとき、サブクラスはprivateで修飾されている以外のスーパークラスのメンバにアクセスできます。設問のコードであれば、Aクラスを継承したBクラスからは、numフィールドにはアクセスできませんが、printメソッドにはアクセスできます。

以上のことから、設問のコードは正常に動作します。したがって、選択肢**D**が正解です。

25. B

➡ Q17

オーバーロードされた別のコンストラクタを呼び出す方法に関する問題です。

オーバーロードされた別のコンストラクタを呼び出すには**this**を使います。選択肢Eのようにメソッド名でコンストラクタ呼び出しはできないので注意してください。また、選択肢Dはインスタンス自身を表す特別な変数thisを使ってメソッド呼び出しの形式でコンストラクタ呼び出しをしています。このような方法でコンストラクタは呼び出せません。よって、選択肢DとEは誤りです。

オーバーロードされた別のコンストラクタを呼び出すコードは、必ず先頭行になければいけません。そのため、選択肢Aのようにほかの処理をしてから呼び出すことはできません。

オーバーロードされた別のコンストラクタを呼び出すときに、フィールドの値を引数としては使えません。そのため、選択肢Cはコンパイルエラーが発生します。

以上のことから、選択肢**B**が正解です。

26. A

➡ Q18

==演算子の比較対象に関する問題です。

==演算子で比較するのは変数の中身です。オブジェクト型変数の場合、変数の中には参照、つまりインスタンスへのリンクが入っています。そのため、==演算子では、2つの変数の参照先が同じであるかどうかを比較することになります。

フィールドの値は、たとえ初期化するコードが明示的に書かれていなくても、インスタンス生成時にデフォルト値で初期化されます。設問のSampleクラスのvalフィールドは、オブジェクト型の一種であるString型なので、オブジェクト型変数のデフォルト値であるnullで初期化されます。

Mainクラスの4行目のif文では、文字数0のStringオブジェクトであるブランク文字列を使っています。このif文の条件式では、Sampleクラスのインスタンスのvalフィールドの値（null）と、ブランク文字列であるStringインスタンスへの参照先が同じであるかどうか比較しています。文字列リテラルは、プログラムの実行時にStringインスタンスとして自動的に生成されるため、参照先がnullになることはありません。よって、このif文の条件式はfalseを戻し、valフィールドの値はデフォルト値であるnullのままということになります。
以上のことから、選択肢**A**が正解です。

27. A、C、D

➡ Q18

private修飾子に関する問題です。

privateで修飾されたメソッドは、継承関係の有無にかかわらず、同じクラスからしかアクセスできません。したがって、選択肢**A、C、D**が正解です。選択肢BとEは、サブクラスからのアクセスなので誤りです。

28. D

➡ Q19

コマンドライン引数に関する問題です。

コマンドライン引数は、javaコマンド実行時に渡すことができる引数のことです。渡されたコマンドライン引数の値は、String配列に格納され、mainメソッドの引数として参照が渡されます。

設問のjavaコマンドでは、実行したいクラス名に続けて「test」というコマンドライン引数が渡されています。そのため、mainメソッドの引数として渡されるString配列は1つだけ要素を持っていることになります。

設問のコードでは、添字0と1の値をコンソールに出力しようとしていますが、前述のとおり、このString配列は1つしか要素を持っていません。そのため、このコードは実行時に配列の要素外アクセスを表す例外（ArrayIndexOutOfBoundsException）がスローされます。以上のことから、選択肢**D**が正解です。

29. D

➡ Q20

インタフェースの実現と型変換に関する問題です。

インタフェースを実現（implements）したクラスは、インタフェースに宣言されているすべてのメソッドを実装しなければいけません。

設問のコードでは、インタフェースAを定義し、それを実現したBとCという2つのクラスを定義しています。どちらのクラスもインタフェースに宣言したメソッドを正しく実装しています。なお、変数名がAとBとCの各クラスで異なりますが、メソッドのシグニチャが同じであれば、変数名の違いは問題にはなりません。

Mainクラスの3行目では、Bクラスとクラスのインスタンスを生成し、2つのインスタンスへの参照を持った配列を作っています。その後、配列の1つ目の要素、つまりBクラスのインスタンスのtestメソッドを呼び出しています（4

行目)。このとき、引数には3と2が渡されているため、Bクラスのtestメソッドからは「(3 * 2)/2」の結果である6が戻されます。

Mainクラスの5行目では、配列の2つ目の要素、つまりCクラスのインスタンスのtestメソッドを呼び出しています。このときも引数には3と2が渡されているため、Cクラスのtestメソッドでは、まず式「3.2 * (3 * 2)」が実行され、その結果、19.2がint型に変換されることで19(小数点以下切り捨て)になって戻されます。

以上のことから、コンソールには「3 19」が表示されます。よって、選択肢**D**が正解です。

30. D ➡ Q21

キャストに関する問題です。

キャストは、型変換を行う際に、互換性について問題がないことを明示的に記述する「保証」です。プログラマーが、型変換が安全に行えることを明示することで、コンパイラは型変換ができるものだと判断します。

設問のコードのMainクラスでは、Shapeクラスのインスタンスを生成し、Shape型の変数にその参照を代入しています(3行目)。4行目では、Shapeクラスのサブクラスであるクラス型の変数を用意し、Shape型変数が持っている参照を代入しようとしています。このとき、コンパイラはShapeクラスにはTriangleクラスだけが持っている定義(サブクラスとしての差分)を持っていないため、型変換はできないと判断します。しかし、キャストを記述して型変換が安全にできることを明示しているため、コンパイラは問題がないと判断します。そのため、このコードではコンパイルエラーは発生しません。

しかし、実際に実行しようとすると、ShapeクラスのインスタンスはTriangleクラスだけが持っている定義(サブクラスとしての差分)を持っていないため、参照先にあるインスタンスをTriangle型として扱うことができません。そのため、実行時にClassCastExceptionがスローされます。以上のことから、選択肢**D**が正解です。

31. A ➡ Q21

オーバーロードに関する問題です。

オーバーロードとは、引数が異なる同じ名前のメソッドを複数定義する「多重定義」のことです。JVMは、実行するメソッドの判別に、メソッド名と引数(数、型と順番)からなるシグニチャを使います。戻り値型はシグニチャ

には含まれないことに注意しましょう。

A. 同じメソッド名で、引数の数が異なります。
B. メソッド名と引数が同じで戻り値型が異なります。よって、これはオーバーロードしたメソッドではなく、「同名のメソッドが2つある」としてコンパイルエラーになります。
C. メソッド名が異なります。よって、オーバーロードではありません。
D. 戻り値型が定義されていません。よって、コンパイルエラーになります。

したがって、選択肢**A**が正解です。

32. A、B、C、G ➡ Q22

ソースファイルの規則に関する問題です。

パッケージに属するクラスを定義するときには、package宣言をソースファイルの先頭行に記述します。package宣言よりも先にコメント以外の何らかのコードを記述すると、コンパイルエラーになります（選択肢**B**）。

クラスは必ず何らかのパッケージに属さなければいけません。たとえpackage宣言を省略した場合でも、デフォルトパッケージ（無名パッケージ）に属します。クラスがデフォルトパッケージに属する場合は、package宣言を記述しません（選択肢**A**）。

1つのソースファイル内には、クラスやインタフェース、列挙型を複数定義できます。ただし、publicなクラス、インタフェース、列挙型は、1つのファイルに1つしか記述できません。ほかのアクセス修飾子を持つクラスであれば、定義できるクラスやインタフェース、列挙型の数に制限はありません（選択肢**C**）。

一方、選択肢Fは、finalクラスは1つのソースファイルに1つしか定義できないとしています。finalクラスは継承を制限し、サブクラスを定義できないクラスですが、前述のとおり、publicでなければ1つのソースファイルに定義できる数に制限はありません。

import宣言は、完全修飾クラス名で記述することを省略してクラス名だけで記述できるようにするためのものです。Javaのプログラムでは、標準クラスライブラリをはじめ、たくさんのクラスを利用します。その際、完全修飾クラス名で記述すると可読性が著しく下がるため、利用するクラスごとにimport宣言を記述するのが一般的です。アスタリスクで省略することも可能ですが、パッケージが異なれば、パッケージごとにimport宣言をします（選択肢D）。

import宣言は、package宣言のあと、クラス宣言の前に記述しなければいけません（選択肢E）。

ソースファイルの名前は、「クラス名.java」でなければいけません（選択肢**G**）。なお、1つのソースファイルに複数のクラスを宣言した場合は、publicなクラスのクラス名と同じファイル名＋拡張子（.java）で保存しなければいけません。

33. A

➡ Q22

for文の条件式に関する問題です。

for文のカッコ「( )」の中には、初期化式、条件式、反復式の3つを記述します。いずれも省略可能です。なお、条件式を省略すると無限ループになるため、処理の中でbreakを使ってループを抜ける処理を記述するなど、プログラムの制御に注意してください。

設問のコードの条件では、1から表示が始まらなければいけません。しかし、初期化式で変数iの値は0で初期化されているため、そのままでは0から表示されることになります。そのため、繰り返し処理を実行する前に値を1増やさなければいけません。for文は、初期化式→条件式→繰り返し処理→反復式の順に動作するため、1から表示するために、条件式で値を1増やす必要があります。

選択肢CとDは、条件式で変数iの値が5よりも小さいかどうかを判定しているだけで、1増やす処理をしていません。そのため、これらのコードでは0から表示が開始してしまいます。よって、選択肢CとDは誤りです。

設問のコードの条件では、4までの数値を表示しなければいけません。そこで、選択肢AとBのインクリメント演算子の前置と後置の違いに着目します。インクリメント演算子が前置の場合は、変数の値を1増やしてからその値が5よりも小さいかを比較します。そのため、1から表示が始まって、4を表示したあと、条件式で値を1増やすことで条件式がfalseを戻すためにfor文を抜けます。以上のことから、選択肢**A**が正解です。

選択肢Bはインクリメント演算子が後置されています。後置の場合は変数の値をコピーしてから1増やし、コピーした値を戻します。そのため、初回の条件式判定では、変数iの値（0）をコピーし、続いて変数の値を1増やしたのち（0→1）、コピーした値（0）を使って「0 ＜ 5」という条件式として評価します。したがって、コンソールへの表示は1から始まります。その後、繰り返し処理が継続して変数iの値が4になったときに、その値（4）をコピーし、続いて変数の値を1増やしたのち（4→5）、コピーした値（4）を使って「4 ＜ 5」という条件式を評価します。この式はtrueを戻すため、コンソールには変

数iの値である5が表示され、その後の条件式でfor文を抜けます。以上のことから、選択肢Bでは1から5までの数値が表示されることになります。よって、選択肢Bは誤りです。

34. B ➡ Q23

ifを使った分岐処理に関する問題です。

if文は、条件に一致した場合のみ処理をする分岐のための構文です。**if-else if**を使えば、複数の条件を指定することができます。設問では、**if-else if-else文**を使い、2つの条件とその他の計3つの分岐を作っています。

1つ目の条件式では、変数aの値が10でなければtrueを戻します。しかし、変数aの値は10であるため、この条件式の結果はfalseとなります。よって、コンソールにAが表示されることはありません。

2つ目の条件式では、変数aの値が変数bよりも小さければtrueを戻します。変数aの値は10、変数bの値は20であるため、この式はtrueを戻します。そのため、コンソールにはBが表示されます。よって、選択肢**B**が正解で、選択肢Cは誤りです。

switch文の場合は、caseラベルに対応した処理の終わりにbreakの記述を省略した場合、一致したcaseラベル以降のすべての処理が実行されます。しかし、if文ではそのようなことはありません。そのため、選択肢DのようにB、Cの順に表示されることはありません。よって、選択肢Dも誤りです。

35. A ➡ Q23

抽象メソッドとオーバーライドに関する問題です。

抽象クラスに定義した**抽象メソッド**は、そのクラスを継承した具象クラスがメソッドをオーバーライドしなければいけません。抽象メソッドは**abstract**で修飾しますが、オーバーライドした具象メソッドではabstractを外さなければいけません。そのため、選択肢Cは誤りです。

メソッドをオーバーライドするとき、元の定義よりも厳しいアクセス修飾子を付けることはできません。設問のtestメソッドはpublicであり、publicよりも厳しい修飾子で修飾することはできません。選択肢Bは、アクセス修飾子がなく、デフォルトのアクセス修飾子が適用され、パッケージ内からしかアクセスができない指定になっています。よって、選択肢Bは誤りです。

オーバーライドはメソッドの再定義であるため、メソッドのシグニチャは元

の定義と一致させなければいけません。しかし、Sampleクラスのtestメソッドが引数を受け取らないにもかかわらず、選択肢Dのtestメソッドは String型の引数を受け取ります。これではメソッドの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。よって、選択肢Dも誤りです。

以上のことから、選択肢**A**が正解です。

36. D

➡ Q24

デフォルトコンストラクタに関する問題です。

プログラマーがコンストラクタを定義しなかった場合、コンパイラが自動的に追加するコンストラクタが**デフォルトコンストラクタ**です。デフォルトコンストラクタは、引数なしで定義されます。ただし、もしプログラマーが1つでもコンストラクタを定義した場合には、コンパイラが追加することはありません。

設問のSampleクラスはString型の引数を受け取るコンストラクタを定義しています。そのため、デフォルトコンストラクタは追加されません。しかし、Mainクラスの3行目では引数なしのコンストラクタを使ってインスタンスを生成しようとしています。このような引数なしのコンストラクタはSampleクラスに存在しないため、このコードはコンパイルエラーが発生します。したがって、選択肢**D**が正解です。
Sampleクラスに引数なしのコンストラクタの定義を追加すれば、Mainクラスはコンパイル可能となり、実行すると「test」「test」と表示されます。

37. D

➡ Q24

カプセル化、データ隠蔽に関する問題です。

カプセル化したあとに、値が不用意に変更されないようにするためには、データを隠蔽する必要があります。Javaでは、データを隠蔽するためにはフィールドのアクセス修飾子を**private**にします。よって、選択肢AとBは誤りです。publicで修飾すると、どのクラスからでもアクセスができるようになり、いつどのタイミングでどのような値に変更されたかを管理できなくなってしまいます。

abstractは、抽象クラスや抽象メソッドの宣言を修飾するための修飾子です。よって、選択肢Eのようにフィールドをabstractで修飾することはできません。

カプセル化は、インスタンス（オブジェクト）単位で関係するデータと、それを使う処理をまとめることです。staticは、クラス単位で共通の変数を用意

するもので、カプセル化の概念とは関係がありません。よって、選択肢Cは誤りです。

以上のことから、選択肢**D**が正解です。

38. D

➡ Q25

配列型変数に関する問題です。

配列型変数は、配列オブジェクトへの参照を扱うための変数です。Javaでは、配列は複数の値を一度に扱うオブジェクトの一種であるという点に注意しましょう。

設問のコードでは、「2, 4, 6, 8」または「1, 3, 5, 7, 9」という値を持った2つの配列を作り、それぞれarrayとarray2という変数にその配列への参照を代入しています。その後、array2をarrayに代入しているため、この2つの変数はどちらも同じ配列（4行目で生成した配列）への参照を持っていることになります。

以上のことから、コンソールには1、3、5、7、9の順に表示されるので、選択肢**D**が正解です。

39. C

➡ Q25

ローカル変数のスコープに関する問題です。

メソッド内に定義した変数のことを「**ローカル変数**」と呼びます。ローカル変数は、メソッド内で宣言した箇所以降で利用できます。

設問のコードでは、mainメソッドで変数xを宣言しています。この変数はmainメソッドの中だけで有効な変数です。次の行ではtestメソッドを呼び出すときに変数xを渡していますが、これはあくまでも変数xの中の値（10）をコピーして渡すだけであって、変数そのものを渡しているわけではありません。そのため、testメソッド内で宣言していない変数xを使うことはできません。以上のことから、このコードはコンパイルエラーになります。よって、選択肢**C**が正解です。

40. D

➡ Q26

do-while文を使った繰り返し処理に関する問題です。

設問では0から2までの値を順に表示しなければいけません。選択肢AやBのよ

うにwhile文の条件式の中でインクリメントしてしまうと、繰り返し処理の前に条件式が処理され、1から表示が始まってしまいます。よって、選択肢AとBは誤りです。

do-while文は、まず繰り返し処理を実行してから条件式を判定する、後判定の繰り返し構文です。そのためインクリメントは、最初に0が表示し終わったあとに実行されます。do-while文は、次のように記述します。

書式

```
do {
    繰り返し処理
} while ( 条件式 );
```

選択肢Cは、doのすぐ後ろに条件式を記述しています。このコードは構文の誤りとしてコンパイルエラーになります。以上のことから、選択肢Cが誤りで、選択肢**D**が正解となります。

41. C

➡ Q26

for文による繰り返し処理の制御に関する問題です。

for文は、初期化式、条件式、反復式の3つで構成されていますが、設問のコードでは初期化式と反復式の2つが空欄となっています。

配列には1から5までの数値が入っているため、この配列の1つ目の要素から順に表示していけば設問の条件に合致します。配列の要素は0番から始まるため、初期化式では変数iの値を0にしなければいけません。よって、選択肢BとDは誤りです。

次に、反復式は条件式を評価する前の処理をするためのもので、初期化式で宣言した変数の値をインクリメントすることがよく行われます。しかし、繰り返し処理の中でコンソールに値を表示したあと、変数iの値をインクリメントしているため、もし反復式でもインクリメントしてしまうと、2つずつ値が増えることになります。そのため、反復式では何も処理する必要がありません。よって、選択肢**C**が正解です。

42. B

➡ Q27

StringクラスのcharAtメソッドに関する問題です。

charAtメソッドは、文字列の中から指定された位置にある1文字を抽出するメソッドです。charAtメソッドで抽出する文字を見つけるには、次のように文

字の間に線を引き、番号を振ってみるとわかりやすいでしょう。

```
| J | F |M| A |M| J | J | A | S | O | N | D |
0   1   2   3   4   5   6   7   8   9   10  11  12
```

設問のコードでは、charAtメソッドに7を渡しています。そのため、上記の図から7番目の線の次にある「A」が抽出されます。以上のことから、選択肢**B**が正解です。

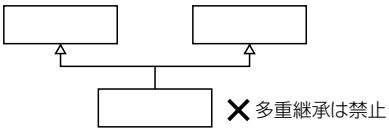
43. B、D

➡ Q27

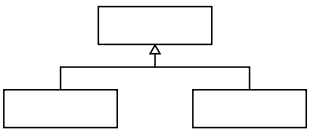
継承に関する問題です。

継承は、あるクラスを拡張した新しいクラスを定義することです。拡張したクラスは、拡張の元となったクラスの特徴を引き継ぎます。ただし、すべてを引き継ぐわけではありません。privateで修飾されたフィールドやメソッド、コンストラクタは引き継ぎません（選択肢C）。

Javaでは、複数のクラスを一度に継承する多重継承は許可されていません。継承は、たいへん便利な機能である一方で、使い方を間違えると変更に弱い設計につながる可能性があるからです（選択肢A）。



このようにJavaでは多重継承が禁止されている一方で、1つのクラスを拡張した複数のクラスを定義することは可能です（選択肢**B**）。



継承したクラス（サブクラス）をさらに拡張したクラスを定義することは可能です（選択肢**D**）。

44. B、D

➡ Q27

修飾子に関する問題です。

abstractは、宣言だけで具体的な処理内容を持たないメソッド、もしくはその具体的な処理内容を持たないメソッドを持つクラスの定義にだけ付けることができる修飾子です。よって、変数やパッケージをabstractで修飾することはできません。

以上のことから、選択肢**B**と**D**が正解です。

45. A

➡ Q28

staticメソッドとインスタンスメソッドの違いに関する問題です。

同一クラスにおいて、staticメソッドからは、staticフィールドかstaticメソッドにしかアクセスできません。一方、インスタンスメソッドからは、staticの有無にかかわらずフィールドやメソッドにアクセスできます。

設問のSampleクラスのgetNumメソッドはstaticメソッドであるため、staticなものにしかアクセスできません。このメソッドがアクセスしているフィールドnumはstaticで修飾されているため、このコードに問題はありません。

また、同じクラスのtestメソッドからgetNumメソッドが呼ばれていますが、前述のとおりインスタンスメソッドからstaticメソッドにアクセスすることは可能です。よって、このコードも問題がなく、Sampleクラスは正常にコンパイルできます（選択肢C）。

Mainクラスのmainメソッドでは、Sampleクラスのインスタンスを生成し、インスタンスへの参照を使ってstaticメソッドとインスタンスメソッドを呼び出しています。しかし、このようなインスタンスへの参照を使ったstaticメソッドの呼び出しは、コンパイル時に次のようにクラス名を使った呼び出しに変更されます。

コンパイル前　`s.getNum()`　➡　コンパイル後　`Sample.getNum()`

そのため、staticメソッドをインスタンスへの参照を使って呼び出しても問題はありません。よって、Mainクラスも正常にコンパイルできます（選択肢D）。

staticフィールドも、インスタンスフィールド同様にデフォルト値で初期化されます。int型のデフォルト値は0であるため、Sampleクラスのnumフィールドも0で初期化されます。

mainメソッドでは、まずSampleクラスのインスタンスのtestメソッドを呼び出しています。このコードでは、さらにgetNumメソッドを呼び出しており、その中でnumフィールドの値を1増やしてから戻します。そのため、このtestメソッドを実行した結果は1が戻されます。

次にSampleクラスのgetNumメソッドを呼び出しています。このコードは前述のとおり、numフィールドの値を1増やしてから戻します。numフィールドの値は先ほどのtestメソッドの呼び出し時に1になっているため、さらに1増やされて2が戻されます。

その結果、変数resultには1+2の結果である3が代入されることになります。以上のことから、コンソールには3が表示されます。よって、選択肢**A**が正解です。

46. D

➡ Q29

インタフェースを使ったポリモーフィズムに関する問題です。

ポリモーフィズムが成り立つのは、継承関係（extends）または実現関係（implements）のどちらかの関係にあるときだけです。

設問のコードでは、Sampleインタフェースを定義し、mainメソッドではそのインタフェース型しか扱わない配列を作っています。また、その配列の要素をTestクラスとExamクラスのインスタンスへの参照で初期化しています。

Testクラスは、Sampleインタフェースを実現しており、Sample型しか扱わない配列の要素として扱うことが可能です。しかし、ExamクラスはSampleインタフェースとは無関係であるため、Sample型しか扱わない配列の要素として扱うことはできません。そのため、このコードはMainクラスをコンパイルする段階でコンパイルエラーが発生します。よって、選択肢**D**が正解です。

47. B、C

➡ Q30

インタフェースのメンバに関する問題です。

インタフェースには、ほかのクラスに公開するためのフィールドとメソッドを宣言できます。よって、**public**以外のアクセス修飾子で修飾することはできません。もし、選択肢Dのようにアクセス修飾子を記述しなかった場合は、コンパイル時にpublicが付加されます。しかし、選択肢Cのように明示的に非公開（private）と記述した場合には、コンパイラによって変更されることはなく、コンパイルエラーが発生します。よって、選択肢**C**はインタフェースには定義できません。

abstractは抽象メソッドか抽象クラスにしか付与できません。インタフェースに宣言できるメソッドは、**抽象メソッド**のみです。そのため、インタフェースに宣言するメソッド宣言には、たとえプログラマーがabstractで修飾を省略しても、コンパイラが自動的にabstractで修飾します。よって、選択肢Dや選択肢Fのようにabstractで修飾していなくても、インタフェースには宣言できます。選択肢Eのように明示的に記述することも可能です。abstractはフィールドには付加できません。よって、選択肢**B**のコードはインタフェースに定義できません。

選択肢Aのようにフィールドを宣言すると、staticとfinalキーワードが自動的に付加され、「`public static final String a = "A"`」という定数となります。インタ フェースには変数は定義できませんが、値が変わらない定数であれば定義できます。よって、選択肢Aは誤りです。

48. B

➡ Q30

インクリメント演算子の実行タイミングとwhile文に関する問題です。

設問のコードでは、変数iの値が4で初期化され、条件式で1増えて5になっています。その後、コンソールにtestという文字を表示し、さらにインクリメントされて値が6になっています。このタイミングでwhile文を抜ければ設問の条件のとおり、testを1回だけ表示できることになります。

「test」と1回表示したあと、条件式に戻るときの変数iの値は6になっています。そのため、選択肢Bのように「6よりも小さければ」という条件式にすれば、while文を抜けることになります。以上のことから、選択肢**B**が正解です。

49. C

➡ Q31

パッケージ宣言とインポート宣言に関する問題です。

パッケージ宣言は、ソースファイルの**先頭行**に記述しなければいけません。パッケージ宣言よりも前に記述できるのは**コメント**だけです。そのため、選択肢BやDのようにパッケージ宣言の前にインポート宣言を記述することはできません。

インポート宣言では、省略表記したいクラスの**完全修飾クラス名**で記述するか、クラス名の部分だけをアスタリスク「*」の**ワイルドカード**で記述するかのどちらかの書式で記述します。

設問の条件では、「com.sample.controllerパッケージに属するクラスを使う」という指定であり、具体的なクラス名は記述されていません。そのため、「com.

sample.controller.*」のようにワイルドカードで記述します。選択肢Aのようにパッケージ名だけを記述しても、具体的にどのようなクラスをインポートするかがわからないため、選択肢Aはコンパイルエラーとなります。以上のことから、選択肢**C**が正解です。

50. D

➡ Q31

二重ループの制御に関する問題です。

二重ループの問題が出題された場合には、内側のループから考えていきます。設問のコードでは、次の部分だけをまず考えます。

例 内側のループ

```
int j = 0;
while( j <= 3 ) {
    System.out.println(++j);
}
```

この部分だけを抜き出してみると、変数jを0で初期化し、前置インクリメントによって値を1つずつ増やしてから表示しています。そのため、コンソールには1から表示されることになります。以上のことから、0から表示が始まっている選択肢AとCは誤りです。

このwhile文の条件式は変数jの値が3以下（3を含む）であるときは繰り返すようになっています。そのため、変数jの値が3のときは繰り返し処理が実行されます。また、表示の前に前置インクリメントによって値が1増えてから表示をするため、コンソールには4が表示されてから、このループを抜けることになります。つまり、この内側のループは1から4までの数値をコンソールに表示することがわかります。

次に外側のループを考えます。内側のループは、次のように結果だけを記述しておくとわかりやすいでしょう。

例 外側のループ

```
for (int i=0; ; i++){
    // 1から4までを表示する
}
```

このループの特徴は、条件式が省略されている点です。このように条件式を省略すると、常に条件に一致していると見なされて無限ループになります。

そのため、設問のコードでは1から4までの値を順番に、かつそれを無限に表示し続けます。以上のことから、選択肢**D**が正解です。

51. A、D ➡ Q32

オーバーロードとオーバーライドの違いに関する問題です。オーバーロードとオーバーライドは混同しやすいので、しっかりと違いを覚えておきましょう。

オーバーロードは、メソッドの多重定義で、引数の異なる同じ名前のメソッドを複数定義することです。一方、**オーバーライド**は、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です。メソッドの再定義であるため、メソッドのシグニチャを変更することはできません。

名前が同じで引数の異なるメソッドを定義するのは、オーバーロードです。オーバーライドは、同じ名前、同じ引数でなくてはいけません。よって、選択肢**A**が正解で、選択肢Bは誤りです。

スーパークラスのメソッドとシグニチャが同じメソッドをサブクラスで定義するのは、オーバーライドです。よって、選択肢Cは誤りで、選択肢**D**が正解です。

52. D ➡ Q32

コンストラクタの特徴に関する問題です。

コンストラクタには次のような3つのルールがあります。

・インスタンス生成時にしか呼び出せない
・戻り値型を記述できない
・コンストラクタ名はクラス名と同じであること

この3つのルールを除けば、コンストラクタは通常のメソッドと変わりません。

コンストラクタは、インスタンスがなければ動作しないインスタンスメソッドの一種です。staticメソッドからインスタンスメソッドへはアクセスできませんが、その逆は可能です。そのため、引数を受け取るコンストラクタからstaticなnumフィールドにアクセスすることは可能です（選択肢A）。

前述のとおり、コンストラクタには戻り値型を記述できません。設問のようにvoid型と記述した場合には、コンストラクタではなく、通常のメソッドとして扱われます。このように戻り値型を記述するとコンストラクタとして扱

われないだけで、文法上の誤りではありません。そのため、コンパイルエラーが起きることはありません（選択肢B）。

ただし、通常のメソッドとして扱われている以上、this()を使ってオーバーロードされた、ほかのコンストラクタを呼び出すことはできません。コンストラクタはインスタンス生成時にしか使えないため、通常のメソッド内から呼び出すことができないからです（選択肢**D**）。

コンストラクタはprivateで修飾することができます。コンストラクタをオーバーロードして、外部から使えるコンストラクタと内部からしか使えないコンストラクタを定義することが可能です（選択肢C）。

53. B、E、F ➡ Q33

配列の宣言方法についての問題です。

3行目は、初期化演算子を使い、配列インスタンスの生成と要素の初期化を同時に行っているコードです。よって、コンパイルエラーは発生しません（選択肢A）。

4行目は、int型の変数にint型しか扱わない配列型変数の値を代入しています。配列型変数の中には、配列インスタンスへの参照が入っています。int型変数には、参照を代入できません。よって、4行目でコンパイルエラーが発生します（選択肢**B**）。

5行目は配列型変数を用意して、3つの要素を持つ配列のインスタンスを生成、その参照を代入しているコードです。このコードでコンパイルエラーは起きません（選択肢C）。

6行目は宣言済みの配列型変数に、新しい配列インスタンスを生成、その参照を代入しています。変数の参照先の配列インスタンスを変更しただけなので、コンパイルエラーは起きません（選択肢D）。

7行目は配列インスタンスの生成をするためにカッコ「( )」を使っています。配列は角カッコ「[ ]」を使わなければいけません。よって、このコードはコンパイルエラーになります（選択肢**E**）。

8行目は配列型変数を宣言しています。配列型変数は、配列インスタンスへの参照が入るだけで、配列そのものを作っているわけではありません。そのため、変数宣言時に要素数を指定することはできません。よって、このコードはコンパイルエラーになります（選択肢**F**）。

54. B

➡ Q34

ポリモーフィズムとオーバーライドに関する問題です。

変数の型は、参照先のインスタンスの扱い方を決めるためのもので、インスタンスの種類を決めるものではありません。

設問のようにA型の変数で扱っても、生成したのはBクラスのインスタンスであり、動作するのもBクラスのインスタンスです。また、サブクラスでスーパークラスのメソッドをオーバーライドした場合、オーバーライドしたメソッドが有効になります。そのため、実行されるhelloメソッドは、Bクラスに定義したhelloメソッドです。よって、コンソールには「B」と表示されます。以上のことから、選択肢**B**が正解です。

55. B、D、E

➡ Q34

抽象クラスに関する問題です。

抽象クラスは、具象メソッドと抽象メソッドの両方を持つことができるクラスです。持つことができるだけで、持たないことも可能です。具象メソッドだけの抽象クラスを定義することは可能です（選択肢A）。

抽象クラスは、インスタンス化できません。そのため、抽象クラスは継承し、具象クラスを定義することを前提としたクラスです（選択肢**B**、選択肢C）。

選択肢**D**や**E**はインタフェースについての説明で、抽象クラスの説明ではありません。よって、説明は誤りです。

56. B、D、F

➡ Q35

命名規則に関する問題です。

クラス名に使えるのはUnicode文字と数字、記号です。使える記号は、ドル記号「$」とアンダースコア「_」の2種類です。また、数字は2文字目から使うことができます。
選択肢A、C、Eのようにシャープ記号「#」やパーセント記号「%」、ハイフン「-」をクラス名に使うことはできません。

以上のことから、選択肢**B**、**D**、**F**が正解です。

57. A、F、G

➡ Q35

メソッドのシグニチャに関する問題です。

メソッドのシグニチャは、メソッド名、引数の型、数、順序によって構成されています（選択肢B、C、D、E）。修飾子や引数の名前、戻り値型は関係ありません（選択肢**A、F、G**）。

58. B

➡ Q35

ポリモーフィズムに関する問題です。
ポリモーフィズムは、異なるインスタンスを同じ型で扱います。同じ型で扱っているものの、実際に動作するインスタンスは異なるため、処理結果が異なるというのがポリモーフィズムの特徴です。

ポリモーフィズムが成り立つには、継承関係にあるか、あるいはインタフェースの実現関係になければいけません。ただし、継承関係にあるだけでは、前述の「処理結果が異なる」ことが実現できません。このように処理結果が異なることを実現するためには、継承した上で、スーパークラスで定義したメソッドをサブクラスでオーバーライドしなければいけません。そのため、選択肢Aよりも選択肢**B**のオーバーライドのほうが、ポリモーフィズムに関係のある用語として適切です。

オーバーロードは、メソッドの多重定義であり、ポリモーフィズムとは関係ありません（選択肢C）。また、インタフェースを継承したインタフェースを定義することもポリモーフィズムとは関係ありません（選択肢D）。

59. C

➡ Q36

インクリメント演算子に関する問題です。

設問のコードでは、変数iを宣言して2で初期化しています。その後、変数の値に2を加算代入しているため、この時点で変数iの値は4になります。この加算代入が終わった段階での式は「`4 + (i++)`」になります。

インクリメント演算子を後置した場合、元の値（4）をコピーしたあとに変数の値を1増やし（5）、その後、先にコピーしておいた値（4）を戻します。そのため、上記の式は「`4 + (4)`」に変更され、コンソールには8が表示されます。よって、選択肢**C**が正解です。

60. C

➡ Q36

オーバーロードに関する問題です。

オーバーロードの条件は、メソッドのシグニチャが異なることです。メソッドのシグニチャは、メソッド名、引数の型、数、順序で構成されています。引数の変数名はシグニチャに含まれないことに注意しましょう。

設問のSampleクラスには、testというメソッドが2つ定義されています。この2つのメソッドは名前が同じだけでなく、引数の型、数、順序も同じです。異なるのは戻り値型だけです。前述のとおり、引数の変数名はシグニチャに含まれないため、これではシグニチャが異なるとは言えず、オーバーロードが成り立ちません。そのため、このコードは同じシグニチャのメソッドを2つ定義しているためコンパイルエラーが発生します。以上のことから、選択肢**C**が正解です。