ITプロ／ITエンジニアのための

徹底攻略

試験番号
1Z0-803

Java SE 7 Silver

［1Z0-803］対応

問題集

志賀澄人 著
株式会社ソキウス・ジャパン 編

インプレスジャパン

本書は、Java SE 7 Programmer Iの受験対策用の教材です。著者、株式会社インプレスジャパンは、本書の使用による同試験への合格を一切保証しません。

本書の内容については正確な記述につとめましたが、著者、株式会社インプレスジャパンは本書の内容に基づく試験の結果にも一切責任を負いません。

インプレスジャパンの書籍ホームページ

書籍の新刊や正誤表など最新情報を随時更新しております。

http://www.impressjapan.jp/

はじめに

本書を手にとっていただき、ありがとうございます。本書は、Oracle Certified Java Programmer, Silver SE 7認定資格を取得するための試験「Java SE 7 Programmer I」（1Z0-803）を受験される方を対象としています。

私は、長年の間、Javaをはじめとしたさまざまな技術を教える仕事をしています。これまでの経験をいかし、「理解するために読む問題集」として、本書を執筆しました。

資格は、取得することも重要ですが、その過程はもっと重要です。有意義な試験対策の時間を過ごしていただきたいという思いから、初心者だからと情報を省略することはしていません。

各章では「理解するための問題と解説」となるよう心がけました。問題は、「ここを押さえておいてほしい」というポイントごとに構成していますので、何が問われているのかを考えながら解くことをお勧めします。また、解説では単に問題に対する説明だけなく、章全体の解説を通して理解が深まるようにしています。ぜひ、解けなかった問題の解説だけでなく、その前後の解説も併せて読んでください。

第9章、第10章の「総仕上げ問題」では、どのような部分に着目すればよいか、どこから着手すればよいかという「解き方」を中心に解説しています。正解を探す「宝探し」のために問題を解いて終わりにするのではなく、理解するために本書を使っていただければ幸いです。

本書が、読者の皆さんのお役に立つことを心から願っています。

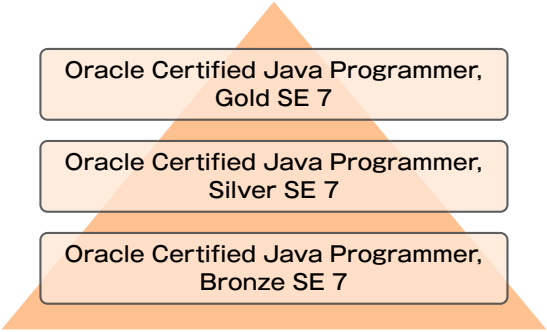
最後に、暖かく手厚いサポートをくださったソキウス・ジャパンの皆さんに、この場をお借りしてお礼申し上げます。

志賀 澄人

Java SE 7 認定資格について

Java SE 7認定資格は、オラクル社がワールドワイドで提供している認定資格で、「Java Platform, Standard Edition 7」に対応しています。

資格は、入門レベルから高度なスキルを証明できるプロフェッショナルレベルまで3段階に分かれています（以下の図を参照）。また、Java SE 6以前のバージョンに対応するJavaプログラマ認定資格の取得者を対象に、Oracle Certified Java Programmer, Gold SE 7（OCJ-P Gold SE 7）へのアップグレードパスが用意されています。



【Java SE 7 認定資格の種類】

認定資格名	説明
OCJ-P Bronze SE 7	初歩的なJavaプログラミングの知識を証明する 「Java SE 7 Bronze」試験の合格が認定の条件となる
OCJ-P Silver SE 7	Javaプログラミング言語を使用した基本的なプログラミングスキルを証明する 「Java SE 7 Programmer I」試験の合格が認定の条件となる
OCJ-P Gold SE 7	Javaプログラミング言語による実践的なプログラミングスキルを証明する 「Upgrade to Java SE 7 Programmer」試験の合格（アップグレード対象）、「Java SE 7 Programmer II」試験の合格（アップグレード対象外）が認定の条件となる

OCJ-P Silver SE 7 について

OCJ-P Silver SE 7は、実務経験1年以上の初級Javaプログラマーを対象としている資格です。OCJ-P Silver SE 7の認定を取得するためには、「Java SE 7 Programmer I」（試験番号：1Z0-803）に合格する必要があります。

試験問題は、ソースコードを参照して、どのような結果になるかを選択するもの

や、ソースコードの誤りを適切に修正する方法など、ソースコードを読解するものが中心です。また、90問の問題を150分で解かなければいけないため、ある程度のスピードでソースコードを読みこなす必要があります。

出題されるトピックは次のようなものです。全体的にまんべんなく押さえておくことが求められます。

- Javaの基本
- Javaのデータ型の操作
- 演算子の決定構造の使用
- 配列の作成と使用
- ループ構造の使用
- メソッドとカプセル化を操作する
- 継承の操作
- 例外の処理

Java SE 7 Programmer I 試験について

本書では、OCJ-P Silver SE 7の試験科目「Java SE 7 Programmer I」を扱います。以下の試験概要は、2014年6月現在の内容です。また、合格ラインや試験料などは今後変更される可能性があります。最新の試験情報は、必ず日本オラクルのWebサイトなどで確認してください。

●試験概要（2014 年 6 月現在）

- 試験名　　:Java SE 7 Programmer I
- 試験番号　　:1Z0-803
- 試験時間　　:150分
- 問題数　　:90問
- 合格ライン　　:63%
- 試験料　　:26,600円（税抜き）
- 試験方法　　:CBTによる選択式

受験申し込み方法

「Java SE 7 Programmer I」試験は、ピアソンVUE社の公認テストセンターで受験可能です。申し込みは、同社のコールセンターまたはWebサイトを利用して行ってください。いずれの場合も希望するテストセンター、日時を選択できます。予約状況によっては選択できない場合もありますので、必ず申し込み時に確認してください。なお、初めてピアソンVUE社に申し込む場合は、同社のWebサイトでアカウント情報を登録する必要があります。

●ピアソン VUE 社

・URL　http://www.pearsonvue.com/japan/IT/oracle_index.html
・TEL　0120-355-583 または 0120-355-173
・FAX　0120-355-163
・Eメール　pvjpreg@pearson.com

●オラクル認定資格に関する問い合わせ先

オラクル認定資格に不明な点がある場合は、オラクル認定資格事務局のメールアドレスに問い合わせることができます。

・Eメール　oraclecert_jp@oracle.com

本書の活用方法

本書は、「Java SE 7 Programmer I」試験の合格を目指す方を対象とした問題集です。本文は、出題範囲に沿った問題と解答で構成されています。第1章から第8章までは、出題範囲のカテゴリ別の章立てになっています。第9章と第10章は、模擬試験の位置付けとなる「総仕上げ問題」です。各章の問題・解説で学習したのちに、実戦形式の総仕上げ問題で受験対策の仕上げをしましょう。

① 問題を解きながら合格レベルの実力が身に付く

単なる試験対策の問題だけでなく、本試験の問題を解くために必要な知識やスキルを身に付けるための問題も掲載されています。第1章～第8章の問題は、解き進めていくとそのカテゴリに関する理解度が深まるように構成されています。

② 丁寧な解説と重要項目がわかる試験対策

解説では、正解・不正解の理由を丁寧に説明しています。また、第1章～第8章の解説ではJavaプログラマーとして必要な知識やスキルを身に付けられるよう、試験対策の解説だけでなく、Javaの技術やオブジェクト指向についてもわかりやすく説明しています。

本文中の「試験対策」欄には、試験の重要項目を挙げていますので、試験対策を効率的に行うことができます。

③ 本試験と同レベルの模擬問題を2回分掲載

第9章と第10章には、本試験と同レベルの問題を掲載しています。受験対策の総仕上げとして、本試験と同じ90問を150分で解いてみましょう。2回分の模擬問題を解くことで、より実戦的な試験対策が可能になります。

本書の構成

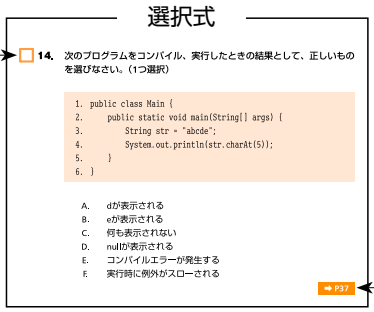
本書は、カテゴリ別に分類された、問題と解答で構成されています。試験の出題範囲に沿った問題に解答したのち、解説を読んで学習すると、合格レベルの実力が身に付きます。また、実際の試験に近い形式になっていますので、より実戦的に学習できます。

問題

試験の出題形式は選択式です。正答を1つだけ選ぶものと複数選ぶものがあります。

チェックボックス

確実に理解している問題のチェックボックスを塗りつぶしながら問題を解き進めると、2回目からは不確かな問題だけを効率的に解くことができます。すべてのチェックボックスが塗りつぶされれば合格は目前です。



解答ページ

問題の右下に、解答ページが表示されています。ランダムに問題を解くときも、解答ページを探すのに手間取ることがありません。

解答

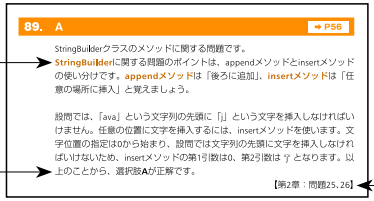
解答には、問題の正解および不正解の理由だけでなく、用語や重要事項などが詳しく解説されています。

重要項目

重要項目は、わかりやすく色文字で示しています。

解説

正解の選択肢は「選択肢**A**」のように太字で示しています。



総仕上げ問題の解説には、各章で詳細に説明している解説への参照先を示しています。復習のために利用してください。

問題のソースコードでは、mainメソッド、例外処理、import文、package文などを省略し、プログラムを動作させるのに不完全な形で記述されているものもあります。このようなソースコードは、実際の試験でも同様に出題されることが考えられます。本書の問題に解答する際には、省略されている記述にかかわらず、問題の趣旨を的確にとらえるようにしてください。

本書で使用するマーク

試験対策	試験対策のために理解しておかなければいけないことや、覚えておかなければいけない重要事項を示しています。	参考	試験対策とは直接関係はありませんが、知っておくと有益な情報を示しています。

目次

第1章

Javaの基本

- クラスの構造
- パッケージ
- クラスのインポート
- staticインポート
- mainメソッド
- javaコマンド

1. 次のうち、クラス宣言に含めることができるものを選びなさい。（3つ選択）

A. メソッド
B. フィールド
C. インポート宣言
D. パッケージ宣言
E. コンストラクタ

➡ P17

2. パッケージに関する説明として、正しいものを選びなさい。（3つ選択）

A. 名前空間を提供する
B. パッケージ名にはドメイン名を逆にしたものを使用しなければならない
C. アクセス制御を提供する
D. クラスの分類を可能にする
E. パッケージに属さないクラスもある

➡ P18

3. 以下の中から、パッケージ宣言が正しく記述されているコードを選びなさい。（1つ選択）

A.
```
import java.io.*;
package aaa;
public class Sample {}
```

B.
```
package aaa;
import java.io.*;
public class Sample {}
```

C.
```
import java.io.*;
package aaa {
    public class Sample {}
}
```

D.
```
import java.io.*;
package aaa (
    public class Sample {}
);
```

➡ P20

4. 次のうち、インポート宣言をしなくても、自動的にインポートされるものはどれか。正しいものを選びなさい。（2つ選択）

A. java.langパッケージに属するクラス
B. java.langパッケージのうち、StringクラスとSystemクラスの2つだけ
C. 同じパッケージに属するクラス
D. サブパッケージに属するクラス

➡ P21

5. 次のクラスをコンパイル、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 Sample {
2.     protected int num = 10;
3. }
```

```
1. package ex5;
2.
3. public class SampleImpl extends Sample {
4.     public static void main(String[] args) {
5.         System.out.println(num);
6.     }
7. }
```

A. 0が表示される
B. 10が表示される
C. コンパイルエラーが発生する
D. 実行時に例外がスローされる

➡ P22

6. 次のプログラムを確認してください。

```
1. public class Sample {
2.     public static int num = 0;
3.     public static void print() {
4.         System.out.println(num);
5.     }
6. }
```

このクラスを利用する、以下のクラスの「// insert code here」に当てはまるコードを選びなさい。（2つ選択）

```
1. // insert code here
2. // insert code here
3.
4. public class Main {
5.     public static void main(String[] args) {
6.         num = 10;
7.         print();
8.     }
9. }
```

A. `import static Sample.num;`
B. `static import Sample.num;`
C. `import static Sample.print;`
D. `import static Sample.print();`
E. `static import Sample.print;`
F. `static import Sample.print();`

➡ P23

7. 次のプログラムを確認してください。

```
1. package ex7;
2.
3. public class Sample {
4.     public final static int VALUE = 100;
5. }
```

このクラスを利用する、以下のクラスをコンパイル、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. package ex7;
2. import static ex7.Sample.VALUE;
3.
4. public class Main {
5.     private final static int VALUE = 0;
6.     public static void main(String[] args) {
7.         System.out.println(VALUE);
8.     }
9. }
```

A. 0が表示される
B. 100が表示される
C. Mainクラスの2行目でコンパイルエラーが発生する
D. Mainクラスの7行目でコンパイルエラーが発生する
E. 実行時に例外がスローされる

➡ P26

8. アプリケーションのエントリーポイントとなるメソッドの条件として、正しいものを選びなさい。（3つ選択）

A. publicであること
B. staticであること
C. 1つのソースファイルに複数記述できる
D. 戻り値型はintであること
E. 引数はString配列型もしくはString型の可変長引数であること
F. 戻り値として0、もしくは1を戻すこと

➡ P27

9. 次のプログラムをコンパイル、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.         System.out.println(args[0] + " " + args[1]);
4.     }
5. }
```

なお、実行時のコマンドは次のとおりとする。

```
> java Main java one two
```

A. 「Main java 」と表示される
B. 「java one 」と表示される
C. 「one two 」と表示される
D. コンパイルエラーが発生する
E. 実行時に例外がスローされる

➡ P28

第 1 章　Javaの基本
解　答

1. A、B、E

➡ P12

Javaのクラス宣言に関する問題です。
クラス宣言は**フィールド**と**メソッド**の2つで構成されており、データを保持するためにどのようなフィールドを持ち、そのフィールドを使ってどのような処理をするかというメソッドを定義します（選択肢**A**、**B**）。**コンストラクタ**はメソッドの一種であるため、メソッド同様にクラス宣言内に記述します（選択肢**E**）。なお、コンストラクタの詳細は第6章の解答12を参照してください。

クラスを定義するときは、インポート宣言やパッケージ宣言も記述しますが、これらはクラス宣言には含まれません。**インポート宣言**や**パッケージ宣言**はソースファイルに対して宣言するものであって、クラスに対して宣言するものではありません（選択肢C、D）。たとえば、次のように1つのソースファイルに2つ以上のクラスを宣言した場合であっても、インポート宣言やパッケージ宣言はどちらのクラスに対しても有効になります。

例 クラスの宣言

```
package aaa;

import java.io.IOException;

class Test {
    void boo() throws IOException {
        throw new IOException();
    }
}

public class Sample {
    void foo() throws IOException {
        throw new IOException();
    }
}
```

このソースファイルでは、TestとSampleの2つのクラスを宣言しています。このソースファイルの先頭でaaaというパッケージを宣言しているため、完全修

飾クラス名はそれぞれaaa.Testとaaa.Sampleになります。また、3行目ではjava.io.IOExceptionクラスをインポートしています。このインポート宣言はパッケージ宣言同様に、宣言されたソースファイルに対して有効です。そのため、同じソースファイルに定義されているTestクラスとSampleクラスのどちらにも有効です。

2. A、C、D

➡ P12

パッケージに関する問題です。
パッケージの目的は、次の3つです。

・名前空間を提供し、名前の衝突を避ける
・アクセス修飾子と組み合わせてアクセス制御機能を提供する
・クラスの分類を可能にする

複雑な要件を満たさなければいけない現代のソフトウェア開発では、すべてのソフトウェア部品を作るわけではありません。過去のプロジェクトで作ったソフトウェア部品を再利用したり、無償で公開されているオープンソースのソフトウェア部品を使ったり、有償で販売されているソフトウェア部品を購入したりすることも頻繁に行われます。このように、現代のソフトウェア開発では、開発生産性を上げるために、他者によって作られたソフトウェア部品を使いながら開発するのが一般的です。

このような開発では、クラス名がほかのソフトウェアのクラス名と重複する可能性が高くなります。もし、名前が重複してしまうとコンパイラやJVMはどちらのクラスを利用してよいか判断できません。その結果、コンパイルエラーが発生したり、設計者が意図したクラスが利用されない事態が発生したりする可能性があります。このような事態を避けるために、コンパイラやJVMは、クラスを「パッケージ名.クラス名」の**完全修飾クラス名**で扱います（選択肢**A**）。

このように、**パッケージ**は名前の重複を避けるために使います。そのため、パッケージ名はできるだけ**一意**なものが推奨されます。そこで、慣習として**ドメイン名を逆にしたもの**がパッケージ名に利用されます。たとえば「xxx.co.jp」というドメインであれば、「jp.co.xxx」という具合です。もちろん、これはあくまでも慣習であって決まりごとではありません。パッケージ名には、ドメイン名以外のものも使用できます（選択肢B）。

クラスを複数のパッケージに分けることで、**パッケージ単位でアクセス制御**ができるようになります。たとえば、次のようなjp.co.xxxパッケージに属する2つのクラスがあるとします。このとき、publicなSampleクラスはほかのパッ

ケージに属するクラスからも使えますが、publicではないTestクラスは使うことができません。

例 jp.co.xxxパッケージに属するpublicなSampleクラスの宣言

```
package jp.co.xxx;
public class Sample {
    // any code
}
```

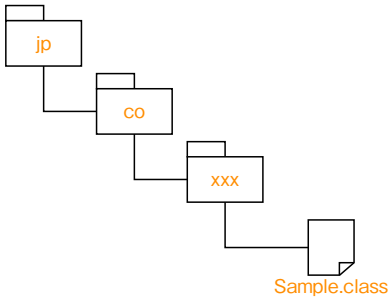
例 jp.co.xxxパッケージに属するTestクラスの宣言

```
package jp.co.xxx;
class Test {   ← publicで宣言していないため外部パッケージからはアクセス不可
    // any code
}
```

このようにパッケージを使ったアクセス制御をすることで、パッケージ内のクラスを「公開するクラス」と「非公開にするクラス」に分類できます。**公開・非公開に分ける**ことで、設計者が意図しないクラスが不用意に使われることを防げます（選択肢**C**）。

パッケージは、**ディレクトリ構造とマッピング**されます。たとえば、「jp.co.xxx.Sample」という完全修飾クラス名を持つクラスは、次のようなディレクトリに配置されます。

【jp.co.xxx.Sampleクラスが配置されるディレクトリの構造】



このようにパッケージとディレクトリ構造がマッピングされると、多数のクラスを分類整理することができ、ソフトウェアの管理が容易になります（選択肢**D**）。

※次ページに続く

クラスは必ず何らかのパッケージに属します。パッケージ宣言を省略したクラスは、デフォルトで**無名パッケージ**に属するものと解釈されます（選択肢E）。パッケージに属さないクラスは存在しません。

3. B ➡ P12

パッケージ宣言に関する問題です。クラスが所属するパッケージは、次のように**packageキーワード**を使って宣言します。

例 パッケージ宣言

```
package sample;  ←ソースコードの先頭行に記述する
public class Test {
    // any code
}
```

このように宣言すると、Testクラスはsampleパッケージに属していることになります。なお、パッケージ宣言は、必ず**ソースコードの先頭行**に記述しなければいけません。間違えやすいのは、次のようにパッケージ宣言とインポート宣言の順番を逆にしてしまうことです。

例 パッケージ宣言（誤り）

```
import aaa.*;
package sample;
public class Test {
    // any code
}
```

このようにパッケージ宣言よりも前に何らかのコードを記述すると、コンパイルエラーが発生します。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。したがって、選択肢Aは誤りで、選択肢**B**が正解です。

ほかのプログラミング言語では、選択肢Cのように中カッコ「{}」を使ってパッケージブロックを作り、そのブロック内にクラスを宣言するものもありますが、Javaでこのような宣言はできません。よって、選択肢CとDは誤りです。

試験対策

パッケージ宣言に関するルールを覚えておきましょう。
・パッケージ宣言は必ずソースコードの先頭行に記述する
・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ある

4. A、C ➡ P13

クラスのインポートに関する問題です。
コンパイラやJVMはクラスを完全修飾クラス名でしか扱えません。パッケージ宣言しなかった場合ですら、そのクラスは**無名パッケージ**（デフォルトパッケージ）に所属していると見なされます。そのため本来は、次のように**完全修飾クラス名**でクラスを指定しなければいけません。

例 完全修飾クラス名でクラスを指定したソースコード

```
public class Main {
    public static void main(String[] args) {
        java.lang.String str = "100";
        int val = java.lang.Integer.parseInt(str);
        java.math.BigDecimal decimal = new java.math.BigDecimal(val);
        System.out.println(decimal.intValue());
    }
}
```

一見してわかるとおり、このように完全修飾クラス名でプログラムを記述すると、コードは冗長で読みにくくなります。そこで、次のように**インポート宣言**をすることで、パッケージ名を省略してクラス名だけで記述できるようにします。

例 クラスのパッケージ名を省略するため、インポート宣言をしたソースコード

```
import java.lang.String;
import java.lang.Integer;
import java.math.BigDecimal;

public class Main {
    public static void main(String[] args) {
        String str = "100";
        int val = Integer.parseInt(str);
        BigDecimal decimal = new BigDecimal(val);
        System.out.println(decimal.intValue());
    }
}
```

※次ページに続く

java.langパッケージは基本的なクラスがまとめられたパッケージであり、このパッケージに所属するクラスは頻繁に利用するためインポート宣言を省略できます。また、同じパッケージに属するクラスのインポート宣言も省略可能です。以上のことから、選択肢**A**と**C**が正解です。選択肢Bはjava.lang.Stringとjava.lang.Systemクラスに限定しており、適切ではありません。
インポート宣言時に、利用するクラスの完全修飾クラス名を指定するのではなく、**アスタリスク**「*」を使って**そのパッケージに属するクラス**をすべてインポートできます。たとえば、次のように宣言すれば、java.utilパッケージ内のすべてのクラスをクラス名で表記できます。

例 java.utilパッケージに属する全クラスのインポート宣言

```
import java.util.*;
```

このようにアスタリスクを使うことで一度に複数のクラスをインポートできます。ただし、この方法でインポートできるのは、**指定したパッケージに属するクラス**に限定されます。アスタリスクを使っても、サブパッケージに属するクラスがインポートされることはありません。上記の宣言でインポートされるのはjava.utilパッケージに属するクラスだけで、サブパッケージであるjava.util.regexパッケージやjava.util.loggingパッケージに属するクラスをインポートできるということではありません。したがって、選択肢Dも誤りです。

5. C ➡ P13

パッケージとクラスのアクセス制御に関する問題です。
無名パッケージに属するクラスは、同じ無名パッケージに属するクラスからしかアクセスできません。たとえば、次の2つのクラスは同じパッケージ（無名パッケージ）に属しているため、OfficeクラスからPersonクラスを使うことができます。

例 同じ無名パッケージに属するPersonクラスとOfficeクラス

```
class Person {}

public class Office {
    Person p;
}
```

しかし、次のように明示的にパッケージ宣言をしたクラスからは、上記のOfficeクラスを使えません。このコードはコンパイルエラーになります。

例 ex5パッケージに属するMainクラス

```
package ex5;

public class Main {
    public static void main(String[] args) {
        Office office = new Office();
    }
}
```

設問のコードでは、Sampleクラスを継承したSampleImplクラスを定義しています。SampleImplでは、Sampleに定義されたnumフィールドの値を表示しようとしています。
しかし、設問のコードの場合、Sampleクラスはパッケージ宣言されていないので無名パッケージに属しており、一方のSampleクラスではex5パッケージに属していることに着目しましょう。明示的にパッケージ宣言したクラスから、無名パッケージに属するクラスにアクセスしようとするとコンパイルエラーになります。以上のことから、選択肢**C**が正解です。

6. A、C

➡ P14

staticインポートの書式に関する出題です。
本来、**staticなフィールドやメソッド**は、次のように「クラス名.フィールド名」や「クラス名.メソッド名」の書式で、どのクラスに定義されているものなのかを明示しなければいけません。

例 staticなフィールドへのアクセスとstaticなメソッドの呼び出し

```
Sample.num          ←staticなフィールドにアクセス
Sample.print();     ←staticなメソッドの呼び出し
```

これをフィールド名やメソッド名だけで省略表記できるようにするための宣言が**staticインポート**です。staticインポートの宣言は、次のように**import static**に続けて、省略表記したいフィールドやメソッドの完全修飾クラス名を記述します。

例 staticインポート宣言

```
import static jp.co.xxx.Sample.num    ←Sampleクラスのstaticなフィールドnumをインポート
import static jp.co.xxx.Sample.print  ←Sampleクラスのstaticなメソッドprintをインポート
```

※次ページに続く

staticインポートを宣言すれば、次のようにあたかも同じクラス内に定義されているフィールドやメソッドのように記述できます。

例 staticインポートを宣言し、フィールドとメソッドを省略表記した例

```
import static jp.co.xxx.Sample.num
import static jp.co.xxx.Sample.print

public class Main {
    public static void main(String[] args) {
        num = 10;
        print();
    }
}
```

以上のことから、選択肢**A**と**C**が正解です。

staticメソッドをインポート宣言するときには、**メソッド名だけ**を記述します。カッコ「**()**」や引数などは記述しません。もし、オーバーロードされた同名のメソッドが複数あった場合はオーバーロードのルールが適用され、渡す引数の型や種類、数によって、呼び出されるメソッドが決まるため、メソッドの数だけstaticインポートを宣言したり、引数を指定したりする必要はありません。

例 staticメソッドを持つTestクラス

```
package ex6;
public class Test {
    public static void print() {
        System.out.println("default");
    }
    public static void print(String str) {
        System.out.println(str);
    }
}
```

例 TestクラスのprintメソッドをstaticインポートするoverloadImportクラス

```
package ex6;
import static ex6.Test.print;
public class OverloadImport {
    public static void main(String[] args) {
        print("sample");
    }
}
```

上記のコードでは、呼び出し時の引数として文字列を渡しているため、String型を受け取るprintメソッドが実行されます。

なお、staticインポート宣言は、インポート宣言同様、省略表記のために使います。インポートしたフィールドやメソッドの定義がコピーされるわけではありません。staticインポートしたフィールドやメソッドは、コンパイル時にstaticインポート宣言に従って、次のように「**クラス名.フィールド名**」や「**クラス名.メソッド名**」の形式に変換され、コンパイルされます。

例 コンパイル時に変換されたフィールドとメソッド

```
public class Main {
    public static void main(String[] args) {
        jp.co.xxx.Sample.num = 10;   ← コンパイラによって変換されたコード
        jp.co.xxx.Sample.print();    ← コンパイラによって変換されたコード
    }
}
```

staticインポートは「import static」であることを忘れないようにしましょう。「staticインポート」という言い方であるため、「static import」のようにimportとstaticの順番を間違えやすいので気を付けましょう。

メソッドのstaticインポート宣言では、メソッド名にカッコ「()」や引数を付けません。

7. A ➡ P15

staticインポートに関する問題です。
staticインポートを多用すると、フィールドやメソッドが重複する可能性が高まります。設問では、Mainクラス内に定義しているstaticフィールドと同名のstaticフィールドであるVALUEをインポートしています。インポートしたクラスに、インポートされたメソッドやフィールドと同名のものがあった場合、そのインポートは無視され、コンパイラによって次のようにコードが変換されます。

例 「クラス名.フィールド名」の形式に変換されたコード

```
public class Main {
    private final static int VALUE = 0;
    public static void main(String[] args) {
        System.out.println(ex7.Main.VALUE);   ← 「クラス名.フィールド名」の形式に変換
    }
}
```

したがって、設問のコードの7行目では、Mainクラス内に定義しているVALUEを使うことになります。以上のことから、選択肢**A**が正解です。

インポートしたクラスに、インポートされたメソッドやフィールドと同名のものがあった場合、そのインポートは無視されます。

なお、同名のフィールドやメソッドを複数インポートした場合、コンパイラはどちらを利用してよいかを判断できません。たとえば、Integerクラスと Longクラスには、staticな定数MAX_VALUEがあります。これを次のように同時にstaticインポートしようとすると、コンパイルエラーが発生します。

例 複数の同名の定数を同時にstaticインポートした例

```
import static java.lang.Integer.MAX_VALUE;
import static java.lang.Long.MAX_VALUE;

public class AnbiguousImport {
    public static void main(String[] args) {
        System.out.println(MAX_VALUE);
    }
}
```

例 上記コードのコンパイルエラー

```
> javac AnbiguousImport.java
AnbiguousImport.java:1: エラー: MAX_VALUEはstaticの単一の型インポート宣言で定義されています
import static java.lang.Integer.MAX_VALUE;
^
エラー1個
```

8. A、B、E ➡ P15

mainメソッドに関する問題です。
クラスには複数のメソッドを定義できます。このとき、どのメソッドから処理を始めるのかが決まっていなくてはいけません。処理を始めるためのメソッドのことを、一般的に**エントリーポイント**と呼びます。JVMは、Javaコマンドで指定されたクラスを読み込み、そのクラスに定義されているエントリーポイントから処理を始めます。

Javaでは、エントリーポイントとなるメソッドの定義は決められており、プログラマーが自由に決められるわけではありません。エントリーポイントは、次のように記述します。

例 エントリーポイントとなるメソッドの定義

```
public static void main(String[] args) {
    // any code
}
```

上記のコード例のうち、変更可能なのは引数名「args」の部分だけで、その他の部分については変更できません。引数名の部分は単なる変数名の宣言に過ぎないため、命名規則に従っていれば自由に変更可能です。エントリーポイントに適用されるルールは次のとおりです（選択肢**A**、**B**）。

- 公開されていること（**public**であること）
- インスタンスを生成しなくても実行できること（**static**であること）
- 戻り値は戻せない（**void**であること）
- メソッド名は**main**であること
- 引数は**String配列型**を1つ受け取ること

エントリーポイントの引数には、String配列型だけでなく、次のように**可変長引数のString型**を受け取ることもできます（選択肢**E**）。

※次ページに続く

```
public static void main(String... args) {

}
```

これは、可変長の引数はコンパイル時に配列型の引数に変換されるためです。以上のことから、選択肢**A**、**B**、**E**が正解です。

9. B ➡ P16

javaコマンドの実行に関する問題です。**javaコマンド**は、JVMを起動するためのコマンドです。JVMは起動後、指定されたクラスをロードし、このクラスのmainメソッドを呼び出します。javaコマンドの構文は次のとおりです。

構文

```
java 完全修飾クラス名［引数 引数 …］
```

クラス名のあとに続ける引数のことを「**起動パラメータ**」や「**コマンドライン引数**」と呼びます。起動パラメータは、スペースで区切って複数指定できます。また、起動パラメータはオプションなので省略可能です。起動パラメータとして指定されたデータは、JVMによってString配列型オブジェクトに格納され、mainメソッドの引数として渡されます。javaコマンドを実行したときの動作は次のとおりです。

- JVMを起動する
- 指定されたクラスをクラスパスから探し出してロードする
- String配列型オブジェクトを作成し、起動パラメータを格納する
- 起動パラメータを保持したString配列型オブジェクトへの参照を引数に渡してmainメソッドを実行する

設問のコマンドでは、「Main」が実行したいクラス名で、そのあとに続く「java」「one」「two」の3つの文字列が起動パラメータです。これらの3つの文字列は、mainメソッドの引数である配列型変数argsを使って参照できます。たとえば、args[0]とすれば「java」、args[1]であれば「one」が参照できます。そのため、設問のクラスの3行目では「java one」とコンソールに表示されます。以上のことから、選択肢**B**が正解です。

試験対策

javaコマンドでのクラス実行時に指定する起動パラメータは、String配列型オブジェクトに格納されるため、1番目が配列型変数args[0]、2番目がargs[1]……となります。

第2章

Javaのデータ型の操作

- プリミティブ型のデータ
- 参照型のデータ
- クラスとインスタンスの概念
- インスタンスフィールドへのアクセス
- インスタンスのメソッド呼び出し
- ガーベッジコレクション
- Stringクラス、Stringクラスのメソッド
- StringBuilderクラス、StringBuilderクラスのメソッド

1. 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.          int val = 7;
4.          bool flg = true;
5.          if (flg == true) {
6.              do {
7.                  System.out.println(val);
8.              } while (val > 10);
9.          }
10.     }
11. }
```

A. 7が1回だけ表示される
B. 何も表示されない
C. コンパイルエラーが発生する
D. 実行時に例外がスローされる

➡ P47

2. 次の中から、コンパイルエラーになる文を選びなさい。(1つ選択)

A. `int a = 267;`
B. `int b = 0413;`
C. `int c = 0x10B;`
D. `int d = 0b100001011;`
E. `int e = 0827;`

➡ P48

3. 次の中から、コンパイルエラーになる式を選びなさい。(5つ選択)

A. `int a = 123_456_789;`
B. `int b = 5________2;`
C. `int c = _123_456_789;`
D. `int d = 123_456_789_;`
E. `float e = 3_.1415F;`
F. `long f = 999_99_9999_L;`
G. `byte g = 0b0_1;`

H. `int h = 0_52;`
I. `int i = 0x_52;`

➡ P49

4. char型の変数の初期化として、正しいものを選びなさい。(1つ選択)

A. `char a = "a";`
B. `char b = 'abc';`
C. `char c = 89;`
D. `char d = null;`

➡ P50

5. 次の中から、コンパイルエラーになる式を選びなさい。(2つ選択)

A. `int $a = 100;`
B. `int b_ = 200;`
C. `int _0 = 300;`
D. `int ${d} = 400;`
E. `int £a = 500;`
F. `int ¥f = 600;`
G. `int g.a = 700;`

➡ P52

6. 次のプログラムを実行した結果としてコンソールに「NULL」と表示したい。3行目の空欄に入るコードとして、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.         [          ]
4.         System.out.println(obj);
5.     }
6. }
```

A. `Object obj = null;`
B. `Object obj = false;`
C. `Object obj = NULL;`
D. `Object obj = "";`
E. 選択肢AとCのどちらも可能である
F. すべての選択肢が誤りである

➡ P53

7. 次のプログラムを確認してください。

```
1. public class Item {
2.     private int num = 10;
3.     public void setNum(int num) {
4.         this.num = num;
5.     }
6.     public int getNum() {
7.         return this.num;
8.     }
9. }
```

このクラスを利用する、以下のプログラムをコンパイル、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.         Item a = new Item();
4.         Item b = new Item();
5.         b.setNum(20);
6.         System.out.println(a.getNum());
7.     }
8. }
```

A. 0が表示される
B. 10が表示される
C. 20が表示される
D. コンパイルエラーが発生する
E. 実行時に例外がスローされる

➡ P54

8. 次のプログラムを確認してください。

```
1. public class Item {
2.     public String name;
3.     public int price;
4.     public void printInfo() {
5.         System.out.println(name + ", " + price);
6.     }
7. }
```

このクラスを利用する、以下のプログラムをコンパイル、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.         Item a = new Item();
4.         Item b = new Item();
5.         a.name = "apple";
6.         b.price = 100;
7.         a.price = 200;
8.         b.name = "banana";
9.         a = b;
10.        a.printInfo();
11.    }
12. }
```

A. 「apple, 100」と表示される
B. 「banana, 100」と表示される
C. 「apple, 200」と表示される
D. 「banana, 200」と表示される
E. 実行時に例外がスローされる
F. コンパイルエラーが発生する

➡ P55

9. 次のクラスのhelloメソッドを呼び出し、コンソールに「hello」と表示したい。

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

4行目の空欄に入るコードとして、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.         Sample sample = new Sample();
4.         [                    ]
5.     }
6. }
```

A. `hello;`
B. `hello();`
C. `Sample.hello;`
D. `Sample.hello();`
E. `sample.hello();`
F. `sample.hello;`

➡ P56

10. 次のプログラムを確認してください。

```
1. public class Sample {
2.     public int add(Integer a, Integer b) {
3.         return a + b;
4.     }
5. }
```

このクラスを利用する、以下のプログラムをコンパイル、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.         Sample s = new Sample();
4.         System.out.println(s.add(10));
5.     }
6. }
```

A. 「10」と表示される
B. 「10null」と表示される
C. 「void」と表示される
D. コンパイルエラーが発生する
E. 実行時に例外がスローされる

➡ P58

11. **次のプログラムを実行し、7行目が終了したときにガーベッジコレクションの対象となるインスタンスはどれか。正しい説明を選びなさい。（1つ選択）**

```
1. public class Main {
2.     public static void main(String[] args) {
3.         Object a = new Object();
4.         Object b = new Object();
5.         Object c = a;
6.         a = null;
7.         b = null;
8.         // more code
9.     }
10. }
```

A. 3行目で作成したインスタンスだけが、ガーベッジコレクションの対象となる
B. 4行目で作成したインスタンスだけが、ガーベッジコレクションの対象となる
C. 3行目と4行目で作成したインスタンスが、ガーベッジコレクションの対象となる
D. ガーベッジコレクションの対象となるインスタンスは存在しない

➡ P58

12. Stringオブジェクトを作成するコードとして、正しいものを選びなさい。（2つ選択）

A.
```
String a = new String("sample");
```
B.
```
String b = "sample";
```
C.
```
String c = String.newInstance("sample");
```
D.
```
String d = String.valueOf('sample');
```

➡ P61

13. 次のプログラムをコンパイル、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.         String str = "hoge, world.";
4.         hello(str);
5.         System.out.println(str);
6.     }
7.     private static void hello(String msg) {
8.         msg.replaceAll("hoge", "hello");
9.     }
10. }
```

A. 「hoge, world.」と表示される
B. 「hello, world.」と表示される
C. 「hello」と表示される
D. 「hello, hello.」と表示される
E. コンパイルエラーが発生する
F. 実行時に例外がスローされる

➡ P62

14. 次のプログラムをコンパイル、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.         String str = "abcde";
4.         System.out.println(str.charAt(5));
5.     }
6. }
```

A. dが表示される
B. eが表示される
C. 何も表示されない
D. nullが表示される
E. コンパイルエラーが発生する
F. 実行時に例外がスローされる

➡ P63

15. 次のプログラムをコンパイル、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.         String str = "abcde";
4.         System.out.println(str.indexOf("abcdef"));
5.     }
6. }
```

A. 0が表示される
B. 1が表示される
C. 4が表示される
D. 5が表示される
E. -1が表示される
F. コンパイルエラーが発生する
G. 実行時に例外がスローされる

➡ P64

16. 次のプログラムをコンパイル、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.         String str = "abcde";
4.         System.out.println(str.substring(2, 4));
5.     }
6. }
```

A. 「bcd」と表示される
B. 「cde」と表示される
C. 「bc」と表示される
D. 「cd」と表示される

➡ P65

17. 次のプログラムをコンパイル、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.         String str = " a b c d e ¥t ";
4.         System.out.println("[" + str.trim() + "]");
5.     }
6. }
```

A. [abcde]と表示される
B. [a b c d e]と表示される
C. [a b c d e　]と表示される
D. [a b c d e]と表示される
E. [a b c d e]と表示される

➡ P66

18. 次のプログラムをコンパイル、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.         String str = "aaaa";
4.         System.out.println(str.replace("aa", "b"));
5.     }
6. }
```

A. 「baa」と表示される
B. 「aab」と表示される
C. 「bb」と表示される
D. 「aba」と表示される

➡ P67

19. 次のプログラムをコンパイル、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.         String str = "abcde";
4.         System.out.println(str.charAt(str.length()));
5.     }
6. }
```

A. aが表示される
B. eが表示される
C. 5が表示される
D. -1が表示される
E. コンパイルエラーが発生する
F. 実行時に例外がスローされる

➡ P67

20. 次のプログラムをコンパイル、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.         String str = "abcde";
4.         System.out.println(str.substring(1, 3).startsWith("b"));
5.     }
6. }
```

A. 「true」と表示される
B. 「false」と表示される
C. 「bc」と表示される
D. 「abc」と表示される
E. コンパイルエラーが発生する
F. 実行時に例外がスローされる

➡ P68

21. 次のプログラムをコンパイルし、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.         String str = "a. b. c. d. e";
4.         String[] array = str.split("¥¥w¥¥s");
5.         for (String s : array) {
6.             System.out.print(s);
7.         }
8.     }
9. }
```

A. 「abcde」と表示される
B. 「a.b.c.d.e」と表示される
C. 「a b c d e」と表示される
D. 「a. b. c. d. e」と表示される

➡ P70

22. 次のプログラムをコンパイル、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.         System.out.println(10 + 20 + "30" + 40);
4.     }
5. }
```

A. 100が表示される
B. 10203040が表示される
C. 303040が表示される
D. コンパイルエラーが発生する
E. 実行時に例外がスローされる

➡ P72

23. 次のプログラムをコンパイル、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.         String str = null;
4.         str += "null";
5.         System.out.println(str);
6.     }
7. }
```

A. 「null」と表示される
B. 「nullnull」と表示される
C. 何も表示されない
D. コンパイルエラーが発生する
E. 実行時に例外がスローされる

➡ P74

24. 次のプログラムをコンパイル、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.         StringBuilder sb = new StringBuilder("abcde");
4.         System.out.println(sb.capacity());
5.     }
6. }
```

A. 0が表示される
B. 5が表示される
C. 16が表示される
D. 21が表示される

➡ P75

25. 次のプログラムをコンパイル、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.         StringBuilder sb = new StringBuilder();
4.         sb.append(true);
5.         sb.append(10);
6.         sb.append('a');
7.         sb.append("bcdef", 1, 3);
8.
9.         char[] array = {'h','e','l','l','o'};
10.        sb.append(array);
11.
12.        System.out.println(sb.toString());
13.    }
14. }
```

A. 4行目でコンパイルエラーが発生する
B. 5行目でコンパイルエラーが発生する
C. 6行目でコンパイルエラーが発生する
D. 7行目でコンパイルエラーが発生する
E. 10行目でコンパイルエラーが発生する

F. 「true10acdhello」と表示される
G. 「true10abchello」と表示される
H. 実行時に例外がスローされる

➡ P77

26. 次のプログラムをコンパイル、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.         StringBuilder sb = new StringBuilder("abc");
4.         sb.append("de").insert(2, "g");
5.         System.out.println(sb);
6.     }
7. }
```

A. 「abgcde」と表示される
B. 「agbcde」と表示される
C. 「abcgde」と表示される
D. 「abcdeg」と表示される

➡ P79

27. 次のプログラムをコンパイル、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.          StringBuilder sb = new StringBuilder();
4.          sb.append("a");
5.          sb.insert(1, "b");
6.          sb.append("cde");
7.          sb.delete(1, 2);
8.          System.out.println(sb);
9.      }
10. }
```

A. 「abcde」と表示される
B. 「cde」と表示される
C. 「ade」と表示される
D. 「acde」と表示される
E. 「bcde」と表示される

➡ P80

28. 次のプログラムをコンパイル、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.          StringBuilder sb = new StringBuilder("abcde");
4.          sb.delete(1, 3);
5.          sb.deleteCharAt(2);
6.          System.out.println(sb);
7.      }
8.  }
```

A. 「d」と表示される
B. 「ad」と表示される
C. 「ae」と表示される
D. 「a」と表示される

➡ P81

29. 次のプログラムをコンパイル、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.         StringBuilder sb = new StringBuilder();
4.         sb.append("abcde");
5.         sb.reverse();
6.         sb.replace(1, 3, "a");
7.         System.out.println(sb);
8.     }
9. }
```

A. 「aade」と表示される
B. 「ade」と表示される
C. 「aba」と表示される
D. 「eaba」と表示される

➡ P81

30. 次のプログラムをコンパイル、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.         StringBuilder sb = new StringBuilder();
4.         sb.insert(0, "abcde");
5.         CharSequence seq = sb.subSequence(1, 5);
6.         String str = new StringBuilder(seq).substring(1,3);
7.         System.out.println(str);
8.     }
9. }
```

A. 「cd」と表示される
B. 「de」と表示される
C. 「bc」と表示される
D. 「ab」と表示される

➡ P82

31. 次のプログラムの5行目に記述できるコードとして、正しいものを選びなさい。（2つ選択）

```
1. public class Main {
2.     public static void main(String[] args) {
3.         int a = 1;
4.         int b = 2;
5.         [          ]
6.         int c = b;
7.     }
8. }
```

A. `System.out.println(a);`
B. `System.out.println(b + 2);`
C. `System.out.println(c);`
D. `System.out.println(d);`

➡ P83

第 2 章　Javaのデータ型の操作
解　答

1. C

➡ P30

Javaのプリミティブ型についての問題です。
データ型とは、**データの種類**を表す情報で、プログラムの実行中に**データの扱い方**を指定するために記述します。たとえば、「3」という値を整数として扱う場合と、浮動小数点数として扱う場合とではコンピュータ内部での扱い方が異なります。これは次のように2進数で表記するだけでも、データの表現方法が異なることがわかります。

【数値の2進数表現】

型	ビット表現
int型の3	00000000000000000000000000000011
float型の3.0	01000000010000000000000000000000

また、原則的にプログラムは同じ種類のデータ同士しか演算できません。数値の「3」と数字の「"2"」を足したり、引いたりはできません。そのため、暗黙的に型変換が行われます。Javaではコンパイラが、データ型を見て型変換できるかどうかをチェックします。

Javaには大きく分けて、**プリミティブ型**と**参照型**の2つのデータ型があります。整数や浮動小数点数といった数値、真偽値、文字を扱うのがプリミティブ型です。もう一方の参照型には、オブジェクト型、列挙型、配列型があります。プリミティブ型と参照型の違いは、値そのものを扱うか、インスタンスへの参照（リンク）を扱うかという点です。プリミティブ型は、次のとおりです。

【プリミティブ型】

データ型	値
boolean	true、false
char	16ビットUnicode文字 ￥u0000～￥uFFFF
byte	8ビット整数 -128～127
short	16ビット整数 -32768～32767
int	32ビット整数 -2147483648～2147483647
long	64ビット整数 -9223372036854775808～9223372036854775807
float	32ビット単精度浮動小数点数
double	64ビット倍精度浮動小数点数

設問のコードは、4行目で宣言しているデータ型が「bool」となっているため、コンパイルエラーが発生します。Javaのプログラムでは、真偽値を扱うためのデータ型は「boolean」です。以上のことから、選択肢**C**が正解です。

JavaはCやC++の影響を強く受けたプログラミング言語ですが、C言語で真偽値を表すbool型はJavaには存在しません。ほかのプログラミング言語を学んだことがある方は、Javaで使われている8つのプリミティブ型を正確に覚えておきましょう。

2. E

➡ P30

整数リテラルの記述に関する問題です。

リテラルとは、ソースコード中に記述する値のことです。Javaには、整数、浮動小数点数、真偽、文字の4つのリテラルがあります。Javaのリテラルは、デフォルトでは、整数値であればint型、浮動小数値であればdouble型、真偽値であればboolean型、文字であればchar型のデータとして扱われます。

もし、リテラルをほかのデータ型であることを明示したいときには、「100L」や「3.0F」のように、**long型**であれば「**L**」や「**l**」、**float型**であれば「**F**」や「**f**」といった**接尾辞**を値の後ろに付けます。なお、byteやshortに対応した接尾辞はありません。これは、変数のデータ型によって、int型のリテラルは自動的に型変換が行われるからです。たとえば次の式では、代入演算子の右オペランドはint型のリテラルであり、これに対応する左オペランドはshort型の変数です。そのため、右オペランドのリテラルは、short型に型変換されます。

例 int型のリテラルをshort型に代入

```
short a = 10;
```

整数リテラルの記述は、10進数のほかに8進数や16進数で記述できます。Java SE 7からは2進数でも記述できるようになりました。たとえば10進数で「63」という整数リテラルを**8進数**で記述するには、「077」のように「**0**」を接頭辞として付けます。**16進数**であれば「0x3F」のように「**0x**」を、**2進数**であれば「0b0111111」のように「**0b**」を**接頭辞**として付けます。

設問の選択肢では、まず接頭辞を確認します。選択肢Aの「267」には接頭辞がありません。よって、10進数の整数リテラルです。選択肢Bの「0413」は、0から始まっていることから8進数の整数リテラルであることがわかります。選択肢Cの「0x10B」は、0xから始まっていることから16進数のリテラルです。選択肢Dの「0b100001011」は、0bから始まっていることから2進数のリテラ

ルです。選択肢Eの「0827」は、0から始まっていることから8進数の整数リテラルと解釈されます。しかし、8進数は0〜7の8つの数を使って値を表すため、8という数は使えません。以上のことから、選択肢**E**がコンパイルエラーになります。

Javaでは、数値を10進数のほかに、2進数、8進数、16進数のリテラルで表記でき、それぞれ、0b、0、0xで始めます。

3. C、D、E、F、I

➡ P30

Java SE 7から導入された新しい整数リテラル表記についての問題です。
Java SE 7から導入された**アンダースコア**「_」を使った数値表記は、桁数の多い数値リテラルの見やすさを向上させる目的で導入されました。数値リテラルのアンダースコアは、以下のルールに従えば、出現する場所や回数は基本的には自由です。

- **リテラルの先頭と末尾**には記述できない
- **記号の前後**には記述できない

なお、2つ目のルールの記号には、小数点を表すドット「.」、long型やfloat型リテラルを表す「L」や「F」、2進数を表す「0b」、16進数を表す「0x」などが含まれます。

A. B. G. H　上記の2つのルールに反していません。よって、コンパイルエラーは発生しません。
C. D　1つ目のルールに反しているため、コンパイルエラーが発生します。
E. F. I　2つ目のルールに反しているため、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**、**D**、**E**、**F**、**I**が正解です。

Java SE 7から導入された整数リテラル表記の詳細については、Javaの仕様を確認してください。試験対策としては以下のことを覚えておきましょう。

- リテラルの先頭と末尾には記述できない
- 記号の前後には記述できない
- 利用できる記号は、小数点を表すドット「.」、long型やfloat型リテラルを表す「L」や「F」、2進数を表す「0b」、16進数を表す「0x」

4. C ➡ P31

文字リテラルと文字列リテラルの違いに関する問題です。
char型は文字を表すデータ型です。複数の文字を集めたものを「文字列」と呼びます。文字と文字列は間違えやすいので注意しましょう。Javaでは、文字リテラルと文字列リテラルを分けるために、リテラルを表す記号が異なります。**文字リテラル**は、「'a'」のように**シングルクォーテーション**「'」で括らなければいけません。一方、**文字列リテラル**は、「"abc"」のように**ダブルクォーテーション**「"」で括ります。

選択肢Aは、ダブルクォーテーションで括っているので文字列リテラルです。しかし、文字列リテラルはchar型変数とは互換性がないため、コンパイルエラーが発生します。選択肢Bは、複数の文字からなる文字列をシングルクォーテーションで括っています。しかし、文字列はダブルクォーテーションで括らなければいけません。よって、コンパイルエラーが発生します。以上のことから、選択肢AとBは誤りです。

コンピュータ内部では、文字には番号が振られ、この文字番号によって管理されています。どの文字に何番を割り当て、それをどのようなビットで扱うかを決めたもののことを「文字符号化方式」や「文字コード」と呼びます。日本語を扱える代表的な文字コードには、「Shift_JIS」や「EUC-JP」などがあります。

文字コードは世界中にさまざまな種類がありますが、それぞれに互換性はないことから文字化けなどの問題が発生しました。そこで、1993年に世界中の文字を集めた共通の文字符号化方式「**Unicode**」が作られました。Javaは、このUnicodeを標準の文字コードに採用しています。

Unicodeの文字は、U+の後ろに16進数の数値4桁を付けたコード（U+0000 ～ U+FFFF）で表されます。4桁の16進数は、65,536（16×16×16×16）通りあるため、1つの文字コードでかなりの文字を表現できます。Javaでは、「'¥u30A2'」のように「**¥u**」の接頭辞の後ろに**16進数4桁**を付け、シングルクォーテーションで括って表現します。たとえば、次のコードはコンソールにカタカナの「ア」を表示します。

例 Unicodeの表記

```
char c = '¥u30A2';
System.out.println(c);
```

このように16進数4桁の数値で文字を表現できることから、char型の変数には、**0〜65535**までの数値を代入できます。次の例では、コンソールにアルファベットの「A」を表示します。

例 「A」を数値で表記

```
char c = 65;
System.out.println(c);
```

以上のことから、選択肢**C**が正解です。

char型の変数に代入できるのは、次の3種類です。

- シングルクォーテーションで括った文字（文字リテラル）
- シングルクォーテーションで括った「¥u」から始まるUnicode番号（文字リテラル）
- 0〜65535までの数値（数値リテラル）

選択肢Dのようにnullは代入できません。**null**とはリテラルの一種で、変数が「何も参照しない」ことを表現するためのデータです。プリミティブ型の変数は値を保持するためのものであって、参照を保持できません。よって、nullを代入することはできません。以上のことから、選択肢Dも誤りです。

char型の変数には、ダブルクォーテーション「"」で括った文字列リテラルは代入できません。代入できるのは、シングルクォーテーション「'」で括った文字リテラル、もしくは数値のみです。シングルクォーテーション、ダブルクォーテーションを間違えないようにしましょう。

char型の変数に代入できるのは、0〜65535までの数値だけです。符号付きの整数（負の値）は扱えません。負の整数リテラルを代入しようとするとコンパイルエラーが発生します。

正確には、Unicodeは符号化文字集合です。符号化文字集合とは、扱う文字を集めただけのものです。実際にコンピュータが文字を扱うには、符号化文字集合で定められた文字をどのようにビットに変換するかという符号化方式を決める必要があります。Unicodeの文字符号化方式には、文字を8ビットで表した「UTF-8」や16ビットで表した「UTF-16」などの種類があります。

5. D、G

➡ P31

識別子の命名規則に関する問題です。
Javaでは、変数やメソッド、クラスなどの名前のことを「**識別子**（Identifier）」と呼びます。識別子はプログラマーが自由に決められますが、次のような規則があります。

- **予約語は使えない**
- 使える記号は、**アンダースコア**「_」と**通貨記号**のみ
- **数字から始めてはいけない**（2文字目以降）

Javaでは、プログラムの文を表現するために必要な用語が規定されています。このような用語のことを「**予約語**」や「**キーワード**」と呼びます。予約語には、「int」や「double」といったデータ型を表現するものや、「for」や「if」といった文脈表現するものなどがあります。このようにプログラムを表現するための予約語は、識別子として使えません。予約語と識別子は、明確に区別できなければ、プログラムの解釈ができなくなるからです。

【Javaの予約語】

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

識別子に使える記号はアンダースコアと「¥」「$」「€」「£」などの通貨記号だけで、ハイフン「-」は使えません。間違えやすいので注意しましょう。また、識別子は数字から始めてはいけません。数字は2文字目以降に使えます。

選択肢Aは通貨記号であるドル記号「$」、選択肢BとCは、アンダースコアを使っています。どちらも許容されている記号であり、問題ありません。同様に、選択肢EとFはイギリス£と日本円の通貨記号を使っています。どちらも通貨記号であるため、問題ありません。